

DTIC FILE COPY

4

RADC-TR-89-377
Final Technical Report
February 1990



AD-A220 861

EXPLANATION AND THE THEORY OF EXPERT PROBLEM SOLVING

Ohio State University

Sponsored by
Defense Advanced Research Projects Agency
ARPA Order No. 5309

DTIC
S ELECTE D
APR 17 1990
B

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

Reproduced From
Best Available Copy

90 04 16 258

EXPLANATION AND THE THEORY OF EXPERT PROBLEM SOLVING

B. Chandrasekaran

Contractor: Ohio State University
Contract Number: F30602-85-C-0010
Effective Date of Contract: 16 January 1985
Contract Expiration Date: 31 August 1989
Short Title of Work: Explanation and Theory of
Expert Problem Solving
Program Code Number: 6E20
Period of Work Covered: Jan 85 - Jan 89

Principal Investigator: John Josephson
Phone: (614) 292-0208

RADC Project Engineer: Robert N. Ruberti
Phone: (315) 330-3528

Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Robert N. Ruberti, RADC (COES), Griffiss AFB NY 13441-5700 under Contract F30602-85-C-0010.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADG-TR-89-377		
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A			7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)		
6a. NAME OF PERFORMING ORGANIZATION Ohio State University		6b. OFFICE SYMBOL (if applicable)		7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	
6c. ADDRESS (City, State, and ZIP Code) Lab for Artificial Intelligence Research Dept of Computer & Information Science 1314 Kinnear Rd, Columbus, Franklin Cty OH		8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency		8b. OFFICE SYMBOL (if applicable) ISTO	
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd Arlington VA 22209-2308		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0010			
10. SOURCE OF FUNDING NUMBERS		PROGRAM ELEMENT NO.		PROJECT NO.	
		62301E		E309	
				00	
				01	
11. TITLE (Include Security Classification) EXPLANATION AND THE THEORY OF EXPERT PROBLEM SOLVING					
12. PERSONAL AUTHOR(S) B. Chandrasekaran					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM Jan 85 TO Jan 89		14. DATE OF REPORT (Year, Month, Day) February 1990	
				15. PAGE COUNT 450	
16. SUPPLEMENTARY NOTATION N/A					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Artificial Intelligence		
12	05		Explanation		
			Problem Solving		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This report summarizes the activities and results of a four-year project that started in the middle of January 1985. This effort concentrated on the development of technology for providing knowledge-based expert planning and problem-solving systems with explanation capabilities. This technology enables such systems to explain and justify their decisions, recommendations, problem solving strategies and use of problem data so that users of such systems can have confidence in the consultations and advice provided by the expert system. <i>Keywords: information processing, hypotheses, machine understanding. (KR)</i>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Robert N. Ruberti			22b. TELEPHONE (Include Area Code) (315) 330-3528		22c. OFFICE SYMBOL RADG (COES)

Chapter 1

Introduction

This document reports on a four-year project undertaken with sponsorship of DARPA under the Strategic Computing Program. The aim of the project has been to advance the theory of knowledge-based systems, with special emphasis on enhancing the abilities of such systems to explain and justify their reasoning and conclusions. Providing good explanations is important to enable human users to evaluate whether to accept conclusions, and to facilitate debugging of knowledge bases and problem-solving mechanisms.

Our approach to making systems explainable has been based on two main sets of ideas. The first requires knowledge-based systems to be constructed out of a set of "generic-task" building blocks which are specified at a higher level of abstraction than the rule-frame-proposition level which forms the basis of almost all of the current technology. Specifying systems at this higher level makes it possible to generate perspicuous explanations closer to the conceptual level of a system's users and designers. Furthermore, explanations of problem-solving strategies can arise directly from the task-specific architectures that come with the generic tasks. Software-engineering advantages come along with this style of system building as well. The second set of ideas centers on the use of structural and functional models of the underlying domain to provide explanations that a knowledge system's compiled or purpose-tuned knowledge base cannot

support. Structural and functional models are also useful in support of problem-solving activity, allowing for flexibility and breadth of coverage that is difficult or impossible to achieve with compiled knowledge alone.

The detailed elaboration of these ideas and their embodiments in software as languages and tools give rise to a coherent technology for building knowledge-based systems. The present power of this technology extends only as far as the generic tasks that have been so far analyzed, and the representational power of the languages developed for representing structure and function; but that is far enough to directly support certain commonly occurring forms of reasoning central to the abilities to perform routine diagnosis, design, and planning. Building knowledge systems in this way ensures that such systems are in important respects explainable, and that explanations are couched in vocabulary at especially useful levels of abstraction.

Besides direct progress on explanation-generation, we report progress on a number of related fronts in knowledge-based AI. For purposes of exposition we divide the work into six topics: progress on explanation-generation proper, foundations for explainable knowledge systems, analysis of certain specific knowledge-based reasoning tasks, development of system-building tools, applications of the theory and tools to various domains, and foundations of AI. The remainder of this report consists of introductory discussions of each of these topics, and a series of appendices which report some of the technical content.

Appendix A is a summary of Michael C. Tanner's Ph.D. dissertation on justifying diagnostic conclusions. Appendix B is a summary of Anne Keuneke's Ph.D. dissertation on generating causal explanations of diagnostic conclusions. Appendix C is an article by Chandrasekaran, et. al., on explaining control strategies. Appendix D is an explication by B. Chandrasekaran of the main points of the generic task approach with a discussion of its applicability to diagnosis and routine design. Appendix E is a survey paper on the classification task by B. Chandrasekaran and A. Goel. Appendix F is the description of a mechanism for the abductive assembly task by J. Josephson, et. al. Appendix G is an analysis of the design task by B. Chandrasekaran. Appendix H is a brief description of the generic task toolset with a description of some of the explanation-generation and other advantages over other approaches to building knowledge systems. Appendix I describes an application of the generic task approach and tools to an

operator advisor system for a nuclear power plant. Appendix J is a discussion of competing AI paradigms by B. Chandrasekaran. Appendix K is a model of robot perception as layered abduction by J. Josephson. Finally, Appendix I, lists the publications and technical reports that arose wholly or in part out of the project, with abstracts included for the major entries.



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Chapter 2

Explanation

A number of knowledge systems have been built that explain, either as their primary purpose or as a secondary feature. Most authors writing on knowledge systems consider explanation to be an essential feature of such systems, and devote some energy to describing the explanatory capabilities of their particular program. However there is often a conflation of user modeling and user interface issues with the more central issue of how a system's knowledge is used to form the content of explanations. There is even some natural confusion between the explanations given by programs that explain their own decisions, and those given by programs that have the problem-solving goal of explaining events outside themselves. Generating good explanations is important, but doing so will require that certain conceptual clarifications be made.

2.1 What Explanation Is

According to *Webster's Third New International Dictionary* the word "explain" is derived from the Latin *explanare*, "to level, make plain or clear." [13] But, what does it mean to "make plain or clear" in the context of knowledge systems?

And just what is it about a knowledge system that an explanation makes "plain or clear"?

One type of explanation that knowledge systems have generated is, roughly, scientific explanation, i.e., explaining the world. Several problem-solving systems can be characterized abstractly as explaining a set of data. For example, we can view medical diagnosis as the generation of disease hypotheses to explain symptoms (as done by INTERNIST [16]). The Red system generates antibody hypotheses to explain the results of tests performed in the hospital blood bank [15], and DENDRAL hypothesizes molecular structures to explain spectroscopic data [2]. Philosophers of science have proposed logical reconstructions of how theories can be considered explanations of observed phenomena [1, 14]. Schank recently discussed the kinds of explanation people deem acceptable in everyday life, and developed a computational theory of how to generate such explanations [18].

The other major type of explanation seeks to have programs explain their own decisions, that is, to apply system knowledge to (1) help users understand how the system reached its conclusions, (2) help debug the knowledge base and problem-solving behavior, and (3) convince users that the system's conclusions are reasonable.

Typically knowledge systems produce the two kinds of explanation by very different processes. Systems such as Red and Internist, whose task is to explain data, do so by means of a reasoning process that can associate potential explainers with the data they might explain. They explain themselves by "introspection," i.e., by examining their own problem-solving knowledge and their own memory for the problem-solving episode. So while, conceptually, the two major kinds of explanation are similar, operationally, they are very different.

We can separate the explanation-generation problem in knowledge systems into three top-level functions: generating the content, being responsive, and the human-computer interface.

Generating an explanation's basic content. Given user queries about some as-

pect of system's decision making, we need to generate an information structure containing the elements that make up an explanation. An essential element in constructing such an explanation is how knowledge of the problem-solving task comes into play during explanation. Explanation content can be put together in two ways:

By introspecting—That is, by examining a record of its own problem solving activity and picking appropriate traces containing information relating to users' queries, or by retrieving portions of the knowledge base that were used in making the decision and thus can be used to support it. To do this we need to know how a problem solver can comprehend its own problem-solving activity.

By concocting—That is, by producing a justification that does not depend on how the decision was actually made, but that independently makes the decision plausible. Constructing such *ex post facto* justifications or explanations is necessary when problem solvers have no access to records of their own problem solving activity, or when information contained in those records is unnecessary or incomprehensible to users. The explanation argues convincingly that the answer is correct without actually referring to the derivation process, just as a mathematical proof persuades without necessarily representing the process by which a mathematician discovered the theorem. Generating such an explanation is an interesting problem-solving process in its own right (see Wick, et al. [20], for a particular approach to this problem).

Responsiveness—shaping explanations to match user knowledge. It may not be necessary to communicate all of the available explanation content to users. Systems apply knowledge of user goals, state of knowledge, and the dialog structure to filter, shape, and organize the output of the *explanation content* so that explanations respond to user needs.

The human-computer interface. The two preceding functions produce all the information needed conceptually and logically for the required explanation. However, presentation issues remain: specifically, how an appropriate human-computer interface effectively displays and presents information to users. Some explanations are best presented in natural language, and some in pictorial or graphical form (pie charts, for example).

If explanation content is inadequate or inappropriate—no matter how good the-

ories for responsiveness and interface functions are—then correspondingly poor explanations will be presented. Thus generating the correct explanation content is the central problem in explanation generation. Accordingly, we have concentrated on one aspect of this problem; namely, basing the explanation content on introspection of the system's own problem-solving behavior. We can break this down into the following three types of explanation:

Trace-based explanation—Explaining why certain decisions were or were not made. This type of explanation relates portions of the data in a particular case to the knowledge for making specific decisions or choices. (In an earlier paper we called this *type 1* explanation [8].)

Knowledge justification—Explaining elements of the knowledge base. For example, we can justify a system's compiled knowledge by linking it to deep knowledge from which it was derived. (For a discussion of issues related to "deep" and "compiled" knowledge see Chandrasekaran and Mittal [7].) (In our earlier paper this was called *type 2* explanation [8].)

Strategic explanation—Explaining the problem solver's control behavior and problem-solving strategy. (Called *type 3* explanations in our earlier paper [8].)

Typically, trace-based and strategic explanations report on a problem solver's runtime behavior and so (in general) they cannot be precompiled without running into combinatorial problems. In principle, explanation structures for knowledge justification can be attached to the knowledge fragments at the time the knowledge base is put together.

Figure 2.1 outlines these aspects of explanation. We have been working on a functional representation for deep models of devices to produce knowledge justifications, relating the domain knowledge to problem-solving knowledge [19] [6]. The theory of generic problem-solving types is especially suited to building systems that explain their control strategy (strategic explanation above) [5]. See Appendix C. Systems built using this generic-task approach are typically composed of knowledge-level agents with well-defined problem-solving roles. Consequently they can also produce trace-based explanations—how data match

knowledge—based on each agent's memory of its own problem-solving behavior. More details on this can be found in [8] and Appendix C.

2.2 A Framework for Explanation

In the previous section we categorized the explanation of problem-solving activity into three distinct types: trace-based explanation, how data matches local goals; knowledge justification, how knowledge itself can be justified; and strategic explanation, how control strategy can be justified. These three types correspond to structures that need to be examined when constructing explanations. Next we elaborate the description of these types and give some examples. The examples were created for expository purposes and are not meant to represent actual explanations generated by particular systems.

Trace-based explanation relates actual problem solving steps to problem states or to data describing the problem. Appropriate fragments of the system's runtime behavior must be examined. For example, consider the following medical scenario.

User: Why do you say the patient has cholestasis?

System: Because the patient has high blood bilirubin, jaundice, and X-rays suggest an obstruction in the biliary duct.

Or, the following economic scenario,

User: Why do you conclude that a tax cut is appropriate here?

System: Because a tax cut's preconditions are high inflation and trade deficits, and current conditions include those factors.

These explanations exemplify how problem-specific data matched pieces of the knowledge base and how certain conclusions were drawn. It might be possible to conclude cholestasis from different possible combinations of data. Users want to know which data combination was present in this particular problem. This requires keeping a trace of problem-solving behavior, examining it, and constructing an explanation from the trace.

Users not satisfied with this explanation level might ask.

Why does high blood bilirubin indicate cholestasis? Must it occur in conjunction with jaundice?

Or.

Why is a tax cut a good idea for shrinking trade deficits?

Answers do not involve the particular situation at hand. The system is being asked to explain portions of its knowledge base, which is *knowledge justification*. Knowledge justifications explain knowledge base elements. Such explanations will often be based on how the knowledge was derived. At least four ways exist to obtain problem-solving knowledge, each with its corresponding type of explanation.

By being told directly—When knowledge can only be justified by appeal to authority (for example, "Text book, p. 85.").

By generalizing from examples—When, for instance, "68% of the time, when a tax cut was tried, the trade deficit went down," or "the last time a patient had these symptoms it turned out to be AIDS."

By explanatory inference—When, for instance, a system contains the rule that (given certain symptoms) a certain infectious organism should be hypothesized. This can be justified by the knowledge that, if the disease were

present, it would explain the symptoms. Further justification might include discussing the medical history of recognizing the symptom group as a distinct disease and identifying the infecting organism. The explanatory inference that this organism is the cause of symptoms (and therefore explains them) may not actually be encoded in the system—that is, the system may be unable to make the explanatory inference but still be able to solve problems. However to justify the knowledge, the system will need access to the inferential history.

By derivation from deeper domain understanding—When, for instance, "Tax cuts generally encourage savings, stimulate investment, and increase production (which decreases prices, increases exports, makes domestic goods attractive, and thus reduces trade deficits)." If the knowledge base contains knowledge directly relating tax cuts to shrinking trade deficits, such a chain of reasoning is not required to conclude that a tax cut is appropriate. Nevertheless, the system should keep this knowledge available so that it can be used appropriately for explanation.

Some of our work [6] has concentrated on the fourth category: that is, reasoning from deep models. In particular, to derive fragments of diagnostic knowledge, we have developed an approach that uses an agent's understanding of how a device works. Thus a diagnostic knowledge system, by tracing how diagnostic knowledge was derived from an understanding of the structure and function of the device involved, can justify its knowledge.

Consider the following interactions with a medical system,

User: Why didn't you consider portal hypertension in this case?

System: Because I had ruled out circulatory diseases, portal hypertension is a special case of circulatory diseases, and my strategy is not to consider special cases when I have ruled out the general case.

Or an economic planning consultant,

Table 2.1: An Explanation Framework

Type of Explanation	Source	Comments
Trace-based explanation	Problem solver (expert system)	Problem solvers can explain their reasoning if they have access to a trace of the process.
Strategic explanation	Generic Task	Problem solvers can explain their strategy if they are properly connected to their generic task.
Knowledge justification	Functional Representation	Problem solvers can justify their knowledge if they are properly connected to deep models.

User: Why aren't you suggesting increased tariffs as a way of decreasing trade deficits?

System: Because that plan involves political costs. My strategy is to consider politically easier plans first.

Part of what is explained in these examples is the knowledge system's control strategy, which is *strategic explanation*. Strategic explanations can account for certain "why not?" as well as "why" questions. What is needed is an ability to abstract and match portions of the control strategy to the situation. Typically, an actually useful explanation will involve combined trace-based and strategic explanations.

Table 2.1 outlines this explanation framework. Our theory of problem-solving types (which is also a theory of control structure types) applies to strategic explanations.

2.3 Explanation Based on the Problem-Level Task

The three types of explanation described above are all based on the problem solver itself—its knowledge, strategy, and actions. But, there is more to a knowledge system than these, there is the problem-level task the system performs. This is the central idea behind our generic-task work at Ohio State as well as Clancey's heuristic classification [9]—knowledge systems are not merely domain-specific problem solvers, they have a domain independent character as diagnosis problem solvers, designers, classifiers, etc. Users of knowledge systems understand this and expect a program's actions and conclusions to be consistent with its task. Thus their questions of the system will express puzzlement about how the system's knowledge, strategy, behavior, and conclusions relate to the problem-solving task. The system's responses to user questions ought to recognize this. When users ask questions of a system that claims to do diagnosis, they will be seeking assurance that the system's actions and conclusions satisfy the goals of diagnosis. The system should be able to respond appropriately. This idea was pursued in depth in Michael C. Tanner's Ph.D. dissertation, "Explaining Knowledge Systems: Justifying Diagnostic Conclusions", which is summarized in Appendix A of this report.

2.4 Explanation Based on Functional and Structural Models

Sometimes a user will be interested to know what causal processes underwrite a conclusion, whether or not explicit causal reasoning was part of the reasoning leading to the conclusion. Thus the reasonableness of a plan typically depends on assumptions about the causal consequences of plan actions, and the reasonableness of a diagnostic conclusion typically depends on whether there are plausible causal chains connecting the proposed malfunction or disease to the observed symptoms. But, since it is usually not feasible to prestore all possible causal chains in advance of particular new cases, some representation (and associated processes) must be supposed which is capable of generating such a causal story as needed for new circumstances. This issue was pursued by Anne Keuneke in

her Ph.D. work, and a summary of her dissertation entitled "Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions" is included as Appendix B to this report.

2.5 Summary

We began by saying that virtually everybody working on knowledge systems recognizes the importance of explanation, but there are many confusions, the two main ones being:

1. Between *scientific explanation* (where the knowledge system explains the world) and *self-explanation* (where the knowledge system explains its own actions). The main difference between these two is over what events or actions are being explained. In the end, these may turn out not to be very different (the knowledge system is part of the world), but for now the distinction is worth making.
2. Between the *content* of explanations and the *presentation* of explanations. That is, too often fancy human interfaces and natural language systems are mistaken for good explanation facilities and the problem of finding the right content of the explanation is ignored or trivialized.

Out of concern for how knowledge systems generate the content of explanations we distinguished five kinds of explanation:

1. *Trace-based explanation* relates actual problem solving to a problem state or data describing the problem.
2. *Knowledge justification* explains portions of the knowledge base, often based on how the knowledge was derived.
3. *Strategic explanation* relates the system's actions to its control strategy.

4. *Problem-level explanation* relates the system's actions and conclusions to its problem-solving task.
5. *Model-based explanation* generates plausible causal sequences that underwrite a system's conclusions.

Appendices A through C describe some of our work on generating explanations of knowledge-system decisions and conclusions. Appendix A is an extended summary of Michael Tanner's Ph.D. dissertation on the justification of diagnostic conclusions. Appendix B is an extended summary of Anne Keuneke's Ph.D. dissertation on composing causal sequences to explain diagnostic conclusions. Appendix C is a reprint of an article by Chandrasekaran, et. al., on explaining control strategies which appeared in *IEEE Expert*. Additional papers and technical reports on explanation generation can be located by consulting the final appendix listing publications.

Chapter 3

Foundations for Explainable Knowledge Systems

3.1 The Generic Task Approach

Much of our work on making systems explainable has centered on the idea that there are *generic information processing tasks* in terms of which knowledge systems can be analyzed and designed. A generic task is a type of reasoning, functionally specified by its input and output, and characterized by an organization of knowledge and strategy for control of problem-solving especially appropriate for achieving the specified output. Our work has aimed to identify the individual generic tasks, to analyze them properly, and to clarify their relationships to more complex forms of reasoning such as diagnosis and planning.

An explication of the main points of the generic task approach is given in Appendix D which is a reprint of "Generic Tasks as Building Blocks for Knowledge-based Systems: the Diagnosis and Routine Design Examples" which appeared in *Knowledge Engineering Review* in 1989.

3.2 Functional Representation of Devices

Functional and structural models can supply the knowledge to support explanations that compiled or purpose-tuned knowledge cannot provide. For example the knowledge representing an understanding of how some device works can help to explain diagnostic conclusions that are arrived at without the direct use of this knowledge. For some time we have been concerned with how to represent an agent's understanding of how something works, and with how to use such a representation in support of explanation and problem solving. We have developed a language which is meant to represent this sort of understanding. The basic idea is that an agent's "deep knowledge" of how a device works can be represented in a way that shows how an intended *function* is accomplished as a result of the *behaviors* of its components, leading to a series of *states* of the device. Originally designed to model the understanding of how a mechanical device works, the Functional Representation Language has more recently been used to represent how aspects of a plan can be understood; plans may be viewed as abstract devices. An explanation of the Functional Representation Language may be found in [19] and an important use of this language in generating explanations is described in Anne Keuneke's dissertation, summarized in Appendix B. Reports on other work on the Functional Representation Language, and its uses for generating explanations, can be found by consulting the final appendix to this report.

Chapter 4

Analysis of Various Knowledge-based Reasoning Tasks

4.1 Classification

Classification appears to be a ubiquitous information-processing task underlying much of human perceptual and cognitive processing. It has been addressed by three distinct research paradigms: pattern recognition, knowledge-based reasoning, and connectionism. "From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task" by B. Chandrasekaran and Ashok Goel traces the evolution of mechanisms for classification that occurs as the computational complexity of the problem increases from numerical parameter setting schemes, through schemes using intermediate abstractions, then relations between symbols, and finally to complex symbolic structures that explicitly incorporate domain knowledge. This paper appeared in *IEEE Transactions on Systems, Man and Cybernetics* for May/June 1988, and a copy is included as Appendix E to this report.

Classification, especially hierarchical classification has served our group as an important guiding example of a generic task since B. Chandrasekaran first formulated the generic task idea sometime around 1981 [4] . Within the time period of this project we began to clearly distinguish classification problem solving from the related task of "routine recognition" (i.e. concept matching or hypothesis scoring). Also an analysis of the computational complexity of classificatory reasoning was performed, several versions of CSRL, our tool for building classification systems were implemented, the role of classification in diagnosis was clarified, and many classification systems were build for various domains. Progress on these aspect of classification was documented in various publications; see the final appendix for more details. Most of these publications are available as LAIR technical reports.

4.2 Abduction

Various facets of abductive reasoning have been investigated during the course of the project, especially those related to the problem of assembling composite explanatory hypotheses. As the project began a second-generation abductive assembly mechanism had already been constructed in conjunction with the Red project. Subsequently this mechanism was further analyzed and documented, and "A Mechanism for Forming Composite Explanatory Hypotheses" by J. Josephson, B. Chandrasekaran, J. Smith, and M. Tanner is included as Appendix F.

A third-generation abductive assembly mechanism was implemented as a domain-independent abductive assembly engine, called Peirce [17]. Its domain independence was demonstrated by using it in three different systems (under sponsorship from projects for medical AI) As Peirce was being built and tested a family of paper designs of parallel abductive assembly mechanisms was worked out [12, 10]. Together this family of paper systems can be thought of as a fourth-generation abductive assembly mechanism. The problem-solving mechanism implemented by Michael Tanner for his dissertation work took advantage of new insights into how incompatible hypotheses can be effectively handled, and can be thought of as a fifth generation mechanism for the task. The architecture proposed for perception, described in Appendix K, makes use of the 6th generation of the

abductive-assembly mechanism. This latest mechanism has been implemented under industrial sponsorship as part of the Integrated Generic Task Toolset, and is now being debugged and tested in anticipation of using it both for speech recognition and for building diagnostic systems in medical and engineering domains.

A connectionist architecture for abductive assembly was also designed and analyzed, but not implemented [11].

On the face of it the problem of determining the best composite hypothesis for explaining a set of data is combinatorially explosive, yet humans seems sometimes to be able to do it quite rapidly, for example in diagnosis and story understanding. So we analyzed the computational complexity of abductive assembly with the objective of determining those conditions under which the problem is tractable (solvable in polynomial time). What we found [3] is that determining the plausibility and explanatory coverage of hypotheses must be tractable, that there cannot be many incompatibility relationships or cancellation effects between individual hypotheses, and that certain conditions apply to the manner in which the plausibilities of hypotheses are compared. The general conclusion is that under broad classes of realistically occurring circumstances optimal abduction is impossible—no tractable algorithm exists that can be guaranteed in general to find best explanations. Thus something must be given up, and a more naturalistic or satisficing conception of abduction seems to be called for.

4.3 Design

The beginnings of our work on design problem solving predates this particular project; David C. Brown's Ph.D. thesis under B. Chandrasekaran, completed in 1984, provided both a set of critical concepts with which to analyze the task, and the DSPL shell for encoding design systems. DSPL systems accomplish routine design tasks by a strategy of plan selection and refinement. Progress during this period can be described as extending the analysis beyond routine design, and extending the mechanisms beyond plan selection and refinement.

A book was published entitled *Design Problem Solving: Knowledge Structures and Control Strategies* by David C. Brown and B. Chandrasekaran. A draft of Chapter 2 of the book entitled "Design: An Information Processing Level Analysis" is included as Appendix G.

We investigated the use of the Functional Representation Language to represent prestored design cases, and in particular to capture, for each case, knowledge of how a device's functions arise from its structures and the behaviors of its component subdevices. This is needed to support intelligent adaption of old designs to new circumstances. Overall the approach appears to be very promising, and the Functional Representation Language appears to provide the means of encoding important parts of the knowledge for case-based reasoning in automated design. Interested parties are referred to the papers numbered 45 and 58 in the final appendix.

Another line of investigation concerned how to overcome the "rigidity" of DSPL's plan selection and refinement control strategy. The idea is that a flexible design problem solver should be able to integrate different subtasks by using appropriate different methods as applicable, and depending on the knowledge available. For example knowledge might be available in the form of a decomposition of a particular design problem into a set of smaller subproblems, the solutions of which can be composed to solve the larger problem (or knowledge may be available that can be transformed to this form). Alternatively, at a given point in the design process, knowledge might be available of alternative plans for designing some particular type of component, or knowledge might be available of a similar case which almost completely serves the current needs. If a similar case is recalled, useful knowledge might occur in the form of an ability to detect the crucial ways in which the old case fails to satisfy the current needs, and/or in forms that will support the selective adaption of the case to the new circumstances. These investigations have led to DSPL ++, a new design shell to embody more flexible control and to capture the forms of knowledge that support it; DSPL ++ is under development with support from Boeing.

Chapter 5

Development of system-building tools

For each generic task the organization of knowledge and control of problem-solving suggest a knowledge representation language, or generic tool. Each generic tool is a "shell" for building a problem-solving system for the corresponding generic task. The generic tools provide to the expert system designer an advantage over rules, frames, logic, and semantic nets, similar to the advantage that higher-level programming languages provide over assembly languages for the computer programmer.

The "generic task toolset" has been an active project, in one form or another, at Ohio State LAIR for many years. The period of time covered by this project saw the following new implementations (some sponsored by this project and some by others): a new implementation of CSRL (for hierarchical classification) in Interlisp/LOOPS, separating the classification problem-solving from the hypothesis matching problem solving (which got its own shell called HYPER, HYPothesis matchER); a new implementation of DSPL (for routine design) in Interlisp/LOOPS; an implementation of WWHI (What Would Happen If, for a form of compiled predictive inference); a new implementation of the functional representation language FUNC (for representing devices) and its compiler for

automatically generating diagnostic hierarchies; implementations of CSRL and DSPL in KEE; an implementation of PEIRCE (for abductive assembly) in Interlisp/LOOPS; an implementation of the "toolbed" in Common Lisp and CLOS for integrating the separate tools within a common framework; an implementation of RA (Recognition Agent, a successor to HYPER) on the toolbed foundation in Common Lisp and CLOS; a full-featured CSRL on the toolbed; and a PEIRCE on the toolbed realizing most of the sixth-generation abductive assembly architecture.

Most of these tools received heavy student use as well as use on a number of research projects in medical and process control domains. Experience with using the earlier versions was folded into improving the designs of later versions.

An agreement was signed with Battelle Columbus Laboratories for commercial development of the toolset. As of this writing Battelle is marketing a Common Lisp version of CSRL, and a DSPL is under development.

In October 1986 our laboratory hosted and co-sponsored (with AAAI and DARPA) a workshop on High-Level Tools for Knowledge-Based Systems. Many good discussions were held, and it would be nice to believe that knowledge-based systems technology was significantly advanced by the cross-fertilization of the various projects seeking to go beyond rules and frames to higher-order structures.

Appendix H is a brief description of the generic task toolset.

Chapter 6

Applications of the Theory to Various Domains

Close associations between the LAIR and various other laboratories and researchers at Ohio State has provided rich opportunities both for transfer of technology to various application domains, and as a source for realistic problems to use as an empirical basis for advancing the theory. Collaborators at Ohio State on application domains have included the Laboratory for Knowledge-Based Medical Systems, the Applied AI in Engineering Group, individuals in the departments of Chemical Engineering, Nuclear Engineering, Pathology, Industrial and Systems Engineering, Speech and Hearing Science, Linguistics, and Electrical Engineering. Industrial collaborators on applications have included McDonnell Douglas and Boeing. By way of technology transfer to Battelle Columbus Laboratories in the form of our AI tools and by way of temporary employment for AI students, our theories and methods have found their way to additional applications done by Battelle.

Appendix I describes an application of the generic task approach to decision support for operators of nuclear power plants.

Chapter 7

Foundations of AI

7.1 Beyond Generic Tasks

Recently we have been investigating how task-specific architectures can be constructed from more general problem-solving architectures like Soar (from CMU), with advantages for generality and flexibility. A preliminary step has been taken, which is reported in [89-JJ-GTSOAR] (see the final appendix to this report).

7.2 Connectionism

During the course of the project, thinking in AI was challenged by the emergence of connectionism as an alternative research paradigm. Our lab was not immune from the controversy, and in fact a good deal of thinking and discussion went into deciding just what to take from connectionism and what to leave behind. This intellectual ferment culminated in a series of papers, references to them can be found in the final appendix.

7.3 Analysis of competing AI paradigms

AI as a discipline is in a paradigmatic mess. There is no widespread agreement on the essential nature of intelligence, or on the best theoretical framework for advancing the study of it. While *almost all* workers in AI would agree to the central importance of information-processing activities operating on representations, connectionists have proposed analog processes and representations while the more traditional view is that the important processes and representations are discrete. Moreover, much information processing, analog and discrete, cannot reasonably be called "intelligent". Thus information processing may be a necessary condition for intelligence, but is not sufficient for it, and we are left with the question of just what kind of information processing is intelligence?

This question is discussed in B. Chandrasekaran's "What kind of Information Processing is Intelligence? A Perspective on AI Paradigms and a Proposal" due to appear in *Source Book on the Foundations of AI* edited by Partridge and Wilks. A copy is included as Appendix J to this report. Three very different styles of theory-making in AI are contrasted: architectural theories, logical-characterization theories, and functional theories.

7.4 Model of Perception

Going back at least to Plato in ancient Greece there is a long tradition of belief that perception is an active process, opposed to a more usual conception of the process as completely passive. Some in this tradition, most notably Immanuel Kant in the 18th century and C.S. Peirce spanning the 19th and 20th, have held that perception involves some form of inference - presumably unconscious inference. Peirce in particular proposed that the relevant form of inference is abduction, a form of plausible inference distinct from both deduction and induction. Abductions go from data describing something to hypotheses which explain or account for that data. Abduction is thus akin to scientific explanation as described in section 2.1.

During the course of the project progress was made on computational models of abduction, especially "abductive assembly" or the generation and critical acceptance of composite explanatory hypotheses. This progress was described in section 4.2. In addition the idea that perception uses abduction led to a model of perception which is described in "A Layered Abduction Model of Perception: Integrating Bottom-up and Top-down Processing in a Multi-Sense Agent" by John Josephson, which is included as Appendix K to this report. This model attracted collaborators at Ohio State in various scientific disciplines related to speech recognition and understanding, and is the basis for a multi-disciplinary project which has initial support by NSF and DARPA under their joint initiative on image understanding and speech recognition.

7.5 Uncertainty Handling

The recent debate in AI about uncertainty handling has conflated several conceptually distinct issues, including: theories of objective probability, normative decision theories, psychological theories of human reasoning, and effective knowledge system design. The dubious belief that "uncertainty handling" is a fundamental "natural kind" of intelligent activity has led to attempts to devise formalisms which are useful for all problems. In contrast we have argued that the form and significance of uncertainty is particular to the problem-solving context in which it arises, and that different strategies of uncertainty handling are appropriate to the different contexts. This is discussed in "Uncertainty Handling in Expert Systems: Uniform vs. Task-Specific Formalisms" by B. Chandrasekaran and Michael C. Tanner which appeared in *Uncertainty in Artificial Intelligence* edited by L. Kanal and J. Lemmer, Elsevier Science Publishers, 1986.

Bibliography

- [1] Aristotle. Posterior analytics. In R. McKeon, editor, *The Basic Works of Aristotle*, pages 108–186. Random House, New York, NY, 1941.
- [2] B. G. Buchanan and E. A. Feigenbaum. Dendral and Meta-Dendral: Their applications dimension. In B. L. Webber and N. J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 313–322. Tioga, Palo Alto, CA, 1981.
- [3] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. Some results concerning the computational complexity of abduction. To appear in the proceedings of the Conference on Principles of Knowledge Representation and Reasoning. January 1989.
- [4] B. Chandrasekaran. Towards a taxonomy of problem-solving types. *AI Magazine*, Winter/Spring:9–17, 1983.
- [5] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, Fall 1986.
- [6] B. Chandrasekaran, J. Josephson, and A. Keuneke. Functional representations as a basis for generating explanations. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 726–731, Atlanta, GA, October 1986.
- [7] B. Chandrasekaran and S. Mittal. On deep versus compiled approaches to diagnostic problem-solving. *International Journal of Man-Machine Studies*, 19(5):425–436, November 1983.
- [8] B. Chandrasekaran, M. C. Tanner, and J. R. Josephson. Explaining control strategies in problem solving. *IEEE Expert*, 4(1):9–24, Spring 1989.

- [9] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289–350, December 1985.
- [10] Ashok Goel, John R. Josephson, and P. Sadayappan. Concurrency in abductive reasoning. In *Proceedings of the Knowledge-Based Systems Workshop*, pages 86–92, April 1987.
- [11] Ashok Goel, J. Ramanujan, and P. Sadayappan. Towards a 'neural' architecture for abductive reasoning. In *Proceedings of the Second International Conference on Neural Networks*, volume 1, pages 681–688, 1988.
- [12] Ashok Goel, P. Sadayappan, and John R. Josephson. Concurrent synthesis of composite explanatory hypotheses. In *Proceedings of the Seventeenth International Conference on Parallel Processing*, pages 156–160, august 1988.
- [13] P. B. Gove, editor. *Webster's Third New International Dictionary*. Merriam-Webster, Springfield, MA, 1981.
- [14] C. G. Hempel. *Aspects of Scientific Explanation*. The Free Press, New York, 1965.
- [15] J. R. Josephson, B. Chandrasekaran, J. W. Smith, Jr., and M. C. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(3):445–454, May/June 1987.
- [16] H. E. Pople. The formation of composite hypotheses in diagnostic problem solving. In *Proceedings of the 5th IJCAI*, pages 1030–1037, Cambridge, MA, August 22–25, 1977.
- [17] W. F. III Punch, M. C. Tanner, and J. R. Josephson. Design considerations for peirce, a high-level language for hypothesis assembly. In Kamal N. Karna, Kamran Parsaye, and Barry G. Silverman, editors, *Proceedings of Expert Systems in Government Symposium*, pages 279–281. IEEE Computer Society Press, October 1986.
- [18] R. C. Schank. *Explanation Patterns: Understanding Mechanically and Creatively*. Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986.
- [19] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In J. L.

Kolodner and C. K. Riesbeck, editors, *Experience, Memory and Reasoning*, pages 47-73. Erlbaum, Hillsdale, NJ, 1986.

- [20] M. R. Wick, W. B. Thompson, and J. R. Slagle. Knowledge-based explanation. TR 88-24, Computer Science Dept., Univ. of Minn., Minneapolis, MN, March 1988.

Appendix A

Explaining Knowledge Systems: Justifying Diagnostic Conclusions

Explaining Knowledge Systems: Justifying Diagnostic Conclusions*

Michael C. Tanner

July 28, 1989

Abstract

Problem-solving systems should be able to give explanations that relate their actions and conclusions to the logical structure of the task the system performs. That is, the system and its users share an understanding of what the task is. This shared understanding, or shared model, of the task represents the logical structure of the task, i.e., the features that characterize correct answers and correct problem solving. Explanations can then relate the goal-subgoal structure of the problem solver to this shared understanding. In this paper I develop this idea in the context of diagnosis. I present a model of diagnosis and derive from it the questions that a diagnostic system can be asked—the questions that arise solely because the system does diagnosis. I give an architecture for a system that can do diagnosis, show how parts of this system can be identified with parts of the diagnostic task, and from this mapping of architecture to task derive answers to the diagnostic questions. I illustrate these ideas by describing the explanation component of RED. There are many aspects to the problem of explanation, including the problem of how to present explanations to users. But central to any explanation is its content. This paper is about the content of explanations and how the content can be derived from the structure and memory of problem solvers by reference to the logical structure of their problem-solving task.

1 Types of Explanation and Explaining Systems

In Artificial Intelligence, and particularly in knowledge systems, there has been considerable interest in explanation. A number of systems have been built that explain, either as their primary purpose or as a secondary feature. Most authors writing on knowledge systems consider explanation to be an essential part of such systems, and devote some energy to describing the explanatory capabilities of their particular program. However, they often conflate user modeling and user interface issues with

*This work has been supported by the Defense Advanced Research Projects Agency under RADC contract F30602-85-C-0010; and the NIH Heart, Lung and Blood Institute, grant number 1 RO1 HL38776-01.

the more central issue of using a system's knowledge to form the content of explanations. There is even some natural confusion between the explanations given by programs that explain their own decisions and those given by programs that have the problem-solving goal of explaining events outside themselves.

We have sorted out and described these issues in more detail in other papers [9, 36]. For present purposes the interesting type of explanation is that generated by a problem solver about its own behavior and knowledge. We can break this down into the following four types:

Trace-based explanation—Explaining why certain decisions were or were not made.

This explanation relates portions of the data in a particular case to the knowledge for making specific decisions or choices.

Knowledge justification—Explaining the knowledge base elements. For example, we can justify a system's compiled knowledge by linking it to deep knowledge from which it was derived. (For a discussion of issues related to "deep" and "compiled" knowledge see Chandrasekaran and Mittal [8].)

Strategic explanation—Explaining the problem solver's control behavior and problem-solving strategy.

Task-based explanation—Explaining how the system's actions and conclusions relate to the goals of the task the system performs.

Typically, trace-based and strategic explanations involve a problem solver's runtime behavior and so (in general) they cannot be precompiled without running into combinatorial problems. In principle, explanation structures for knowledge justification can be attached to the knowledge fragments at the time we put the knowledge base together. These three types correspond to structures that must be examined when constructing explanations. The fourth type of explanation, task-based explanation, will be the focus of this paper. This type of explanation relates to the problem solver's run-time behavior, but the problem solver does not need task knowledge to solve problems. In the remainder of this section I will elaborate the description of these types and give some examples.

In our lab we have been working on a functional representation for deep models of devices to produce knowledge justifications, relating the domain knowledge to problem-solving knowledge [32]. I will not discuss this here; see Chandrasekaran, et al. [7], for more information. The theory of generic problem-solving types is especially suited to building systems that explain their control strategy (strategic explanation) [6]. More details on explanation in systems built using this *generic-task approach* can be found in Chandrasekaran, et al. [9].

1.1 Trace-Based Explanation

Trace-based explanation relates actual problem solving to a problem state or data describing the problem. This involves examining appropriate fragments of the sys-

tem's runtime behavior. These explanations tell how problem-specific data matched pieces of the knowledge base and how certain conclusions were drawn. This often requires keeping a trace of problem-solving behavior, examining it, and constructing an explanation from the trace. Trace-based explanation is the most common kind of explanation given by knowledge systems. Virtually all systems do this in one form or another because describing what the system did, and its reasons for doing it, is a basic part of explanation. Probably the most widely known of these systems are SHRDLU and MYCIN.

Terry Winograd's SHRDLU [39] was the first system to be explicitly concerned with both problem solving and explanation of problem solving. SHRDLU could solve problems in a blocks world, and explain why, when, and how it took each action. SHRDLU solves problems by setting up a goal-subgoal structure where each goal is an action for the system to take. It can answer "why," "when," and "how" questions only about these actions. SHRDLU's answers refer back to the goal-subgoal structure.

MYCIN is a rule-based consultation system implemented as a backward-chaining rule system [2]. That is, the system sets up the goal of establishing a rule's consequent, the "then" part of the rule, which results in setting up the subgoals of establishing the antecedent, the "if" part. While solving problems, MYCIN asks users for more information when one of the goals cannot be satisfied by information that MYCIN has available. Users, instead of providing answers, can ask "why" or "how." The explanation technique in MYCIN is explicitly based on the assumption that some sort of trace of the program's actions is an effective explanation [14]. Since MYCIN is based on rules that are chained in a goal-subgoal fashion, this trace-type explanation is based on the rules and the goal-subgoal structure. "Why" corresponds to ascent to the supergoal and is answered by giving the rule that uses the requested information in its premise. "How" corresponds to descent to the subgoals and is answered by giving the rule that concludes about the information. This is identical to SHRDLU in presenting goals and subgoals, with the refinement of using rules to express the goal-subgoal structure.

There have been numerous other knowledge systems with explanation facilities. Some have been built with rules, like MYCIN, and give explanations that are essentially identical to MYCIN's explanations. Many systems that are based on other methods still explain by means of a trace. ABEL, for example, reasons using a causal network—but it explains by translating pieces of the net into English, that is, by tracing its actual reasoning [26]. PROSPECTOR [16] is implemented as a Bayesian inference network, but its explanations are the same as MYCIN's with the rules represented as nodes and links in the net. BLAH [37] is a program with explanations based on a study of "natural explanation" [17]. The goal of this study was to analyze the syntactic organization of explanations and develop a grammar that describes justificatory argument. BLAH constructs explanations according to this grammar. This results in explanations that may be easier to understand, because of their form, but the content of the explanations is a trace of BLAH's reasoning—backward-chained production rules (i.e., MYCIN-like). Most other knowledge systems with explanation facilities can be seen to explain by giving a trace of the reasoning, i.e., trace-based

explanation.

1.2 Knowledge Justification

Users may ask a system to explain portions of its knowledge base, i.e., *knowledge justification*. Knowledge justifications explain knowledge base elements. Such explanations will often be based on how the knowledge was derived. There are several ways to obtain problem-solving knowledge, each with its corresponding type of explanation. These include: being told, generalization, explanatory inference, and derivation from deeper domain understanding. The justification of knowledge in knowledge systems has received far less attention than trace-based explanations. For the most part such justifications have been "canned" text included with the rules in the knowledge base at the time the system was built. There is one interesting exception, XPLAIN. In addition, the functional representation under development at OSU's LAIR may provide a way to get interesting justifications of knowledge.

A knowledge system has task-specific goals and problem-solving knowledge that we can view as *compiled* from more general domain knowledge. If the system remembers a trace of the compilation, it can justify system rules in terms of the deeper knowledge. XPLAIN is a program that makes use of this idea to justify knowledge in a knowledge system [35]. XPLAIN is able to use deep knowledge ("domain model") and a representation of problem-solving control strategies ("domain principles") to compile a knowledge system (the "performance system"). ("Compile" means to convert general knowledge into use-specific knowledge.) By keeping a record of this compilation, XPLAIN can justify a rule in the performance system by referring back to the domain model.

Sembugamoorthy and Chandrasekaran introduced the functional representation as a way of understanding devices and showed how it could be used to compile problem solvers of various kinds [32]. They do not specifically address explanation, although later work described by Chandrasekaran, et al. [7], does. In principle, this is similar to XPLAIN—problem-solving knowledge, e.g., diagnostic knowledge, can be justified by tracing its derivation from an understanding of the structure and function of the device.

As I described earlier, there are several ways of deriving knowledge besides deriving it from a deep model, including being told directly, generalizing from examples, and explanatory inference. Many learning programs derive knowledge in these ways but I do not know of any attempt to have such programs use the derivations to justify their knowledge. Many programs have "canned" justifications, i.e., text strings installed at the time the system is built. For example, GUIDON included justification text with each of MYCIN's rules for use as part of a tutoring system, and Davis suggested capturing such information in general as part of building any knowledge system [10, 14].

1.3 Strategic Explanation

The third kind of explanation is explaining the knowledge system's control strategy, i.e., *strategic explanation*. To do this a system needs an ability to abstract and match portions of its control strategy to a given situation. Typically, the actual explanation involves combined trace-based and strategic explanations. Like knowledge justifications, strategic explanations have received little attention in AI. Only two bodies of research have addressed this type of explanation: NEOMYCIN, and associated systems, and generic-task theory.

Clancey noted that knowledge systems typically perform tasks best described at a higher level than a rule base's goal-subgoal level [11]. But MYCIN had explicit representation of rules only, and not of the problem-solving strategy implicitly encoded in rules by system designers; therefore, it could not answer "why" questions that needed to be interpreted strategically. However, if system behavior is *represented* at the task level, it can produce *explanations* at the task level. NEOMYCIN solves the same diagnosis problem as MYCIN, but explicitly represents the diagnostic task. It contains diagnostic operators including "establish hypothesis space" and "explore and refine" that represent the diagnostic strategy and in terms of which it can explain its problem-solving activity [13]. NEOMYCIN implements strategy by encoding it in metarules (i.e., rules that tell what order to invoke the problem-solving rules). It can explain why a particular decision was made at a particular time by referring back to the metarule that invoked the rule in question. So, by using the same kind of explanation facility as MYCIN, namely, displaying the rules used during problem solving, NEOMYCIN can give strategic explanations that describe its higher level goals by using trace-based explanation methods.

Chandrasekaran has developed a theory of problem-solving types called generic tasks [6]. A goal of generic-task theory is to reduce the gap between the implementation level and the intrinsic task level. A concept central to generic tasks is that: (1) input-output behavior (i.e., the function of the task), (2) kinds of knowledge needed to perform the task, and (3) kinds of inference appropriate for the task, are all specified together. Chandrasekaran [6] and Bylander and Chandrasekaran [4] describe the advantages of this approach for system design and knowledge acquisition. For explanation the important contribution of generic tasks is that each problem solving type has a set of appropriate control strategy goals. These goals can be used to explain why particular actions are taken at particular times.

1.4 Task-Based Explanation

The three types of explanation described above are all based on the problem solver itself—its knowledge, strategy, and actions. But, there is more to a knowledge system than these, there is the problem-level task the system performs. This is the central idea behind our generic-task work at Ohio State [6]—knowledge systems are not merely domain-specific problem solvers, they have a domain independent character as diagnosis problem solvers, designers, classifiers, etc. Users of knowledge systems

understand this and expect a program's actions and conclusions to be consistent with its task. Thus their questions of the system will express puzzlement about how the system's knowledge, strategy, behavior, and conclusions relate to the problem-solving task. The system's responses to user questions ought to recognize this. When users ask questions of a system that claims to do diagnosis, they will be seeking assurance that the system's actions and conclusions satisfy the goals of diagnosis. The system should be able to respond appropriately. My main focus in the remainder of this paper will be on task-based explanation.

As far as I know, no one is doing task-level explanations. The closest to it is some work being done by Wick on knowledge-based justification [38]. In knowledge-based justification the reasoning used to solve the problem and the reasoning used to justify it are completely separate. So, for any given problem, the problem solver works until it gets a solution, then the justifier goes to work to convince the user that the solution is correct. There is no necessary relationship between this justificatory argument and the actual reasoning that produced the solution. In Wick's system the explanation is presented using standard rhetorical techniques. This could easily be problem-level explanation if these techniques were augmented to include the goals of the problem-solving task.

2 Diagnosis

Knowledge systems are problem solvers, but they are not *general* problem solvers. That is, a knowledge system is capable of solving problems of a particular *kind*. This is a key intuition behind our generic-task work at Ohio State as well as Clancey's heuristic classification[12]—that knowledge systems are not merely problem solvers with domain-specific knowledge but that they have a domain-independent character as diagnosis problem solvers, designers, data-base reasoners, classifiers, etc. Furthermore, users of knowledge systems understand this and bring to their interactions with a system an expectation that the program's actions and conclusions will be consistent with the task. Thus their questions of the system will express puzzlement about how the system's knowledge, strategy, behavior, and conclusions relate to the problem-solving task. The system's responses to user questions ought to recognize this. What is needed is a shared model of the problem-solving task that expresses the legitimate concerns of the user and that can be used to both interpret the user's questions and provide sensible answers. This shared model is based on the task not the user, i.e., it is not user specific. It is also a model of the problem, not a method for solving the problem.

2.1 A Shared Model of Diagnosis

As an illustration of the problems of not having a shared model, consider the way MYCIN answers WHY questions.

MYCIN: Is the patient's illness with ORGANISM-1 a hospital-acquired infection?

User: WHY?

MYCIN: This will aid in determining the category of ORGANISM-1. (based on [31, p. 347])

Obviously "WHY?" is ambiguous so MYCIN offers this interpretation of the question:

WHY is it important to determine whether or not the infection with ORGANISM-1 was acquired while the patient was hospitalized?

But note that the actual interpretation, based on the answer given, is:

What goals of MYCIN will be served by knowing the answer to this question?

However, I claim that the user is not asking a question about MYCIN's goals but about the goals of MYCIN's problem-solving task of diagnosis. More particularly, the user wants to know how the requested information will help achieve the goals of diagnosis, e.g., does this information bear on rating the plausibility of disease hypotheses or on deciding what can be explained by disease hypotheses?

In AI, diagnosis is typically described as an abduction problem [15, 23, 27, 29, 30]. That is, the task is to find a disease, or set of diseases, that best explains the symptoms. More specifically, a diagnostic conclusion should explain the symptoms, it should be plausible, and it should be significantly better than the alternative explanations. According to this abductive model of diagnosis, a diagnostic conclusion is supported, perhaps implicitly, by the following argument:

- There is a principal complaint, i.e., a focus set of symptoms that sets the diagnostic problem.
- There are a number of diagnostic hypotheses that might explain the principal complaint.
- Some of the diagnostic hypotheses can be ruled out. This can be due to:
 - * Some diagnostic hypotheses are not pertinent since they cannot explain the principal complaint.
 - * Some diagnostic hypotheses are implausible independent of what they might explain.
- The diagnostic conclusion is the best of the plausible hypotheses that are capable of explaining the principal complaint.

Note that this includes the general case in which diagnostic conclusions contain more than one hypothesis. This argument form represents the logical structure of the diagnostic task and is derived from the abductive nature of diagnosis. In particular, it follows from the meaning of "best explanation."

One of the underlying goals of diagnosis is to enable the selection of a treatment. To do this it is necessary to find out what is true about the patient, i.e., what is causing the principal complaint. The argument form above represents an attempt to ensure that the diagnostic conclusion is the true cause. But suppose the diagnostic conclusion turns out to be wrong. What happened to the true answer? That is, why did the true, or correct, answer *not* turn out to be the best explanation? This can only happen if something is wrong with one or more parts of the diagnosis. Based on the diagnosis model, this means that there is something wrong with the principal complaint, or there is something wrong with the list of diagnostic hypotheses, or there was an error in the rule-out process, or an error in rating or choosing the best hypothesis. Thus, the diagnostic conclusion can only be wrong for one or more of the following reasons:

1. There is something wrong with the principal complaint.
 - (a) The principal complaint is not really present or does not need to be explained.
 - (b) The principal complaint is incomplete, there are other things that should be explained by the diagnostic conclusion.
2. The true answer was not on the list of diagnostic hypotheses thought to have the potential of explaining the principal complaint, thus it never was considered.
3. There is an error in the rule out.
 - (a) The true answer was ruled out. This might happen because:
 - i. It was mistakenly thought to be implausible.
 - ii. It was mistakenly thought not to explain the data.
 - (b) The wrong answer (the one given) was not ruled out. This might happen because:
 - i. It was mistakenly thought to be plausible.
 - ii. It was mistakenly thought to explain the data.
4. There is an error in rating the plausible explanations.
 - (a) The wrong answer is rated too high.
 - (b) The true answer is rated too low.

The source of these errors might be found in either missing or faulty knowledge as well as in various problems with the data itself.

Since a question from the user expresses puzzlement about meeting the goals of diagnosis, it can be interpreted as attempting to ensure that the conclusion is correct. Thus, corresponding to each source of potential error there is a class of questions. This analysis tells us that if we build a knowledge system and claim that it does diagnosis, we can expect it to be asked certain questions. It could be asked

the following classes of questions, challenging potential weaknesses in the system's reasoning:

1. Is the principal complaint really present or abnormal? This group includes, "How sure are you that finding f_i is present?" and, "Was finding f_i derived by a method that often shows false positives?"
2. Does the principal complaint contain all the important data? For example, "Should the diagnosis also explain finding f_i ?" and "Did you check for the presence of finding f_i ?"
3. Was a broad enough set of explanatory hypotheses considered? This includes such questions as, "Did you consider hypothesis h_i ?" and, "What hypotheses did you consider?"
4. Has some hypothesis been incorrectly ruled out? This includes, "Why was hypothesis h_i ruled out?" and, "Why doesn't the diagnosis include hypothesis h_i ?"
5. Could some hypothesis explain a finding that the system thought could not? For example, "Why doesn't hypothesis h_i explain finding f_j ?" and, "What are the possible explanations for finding f_j ?"
6. Was some hypothesis not ruled out that should have been? For example, "Why is hypothesis h_i considered plausible even though it is rarely present in cases where finding f_j is found?" and, "Why not rule out hypothesis h_i on the basis of the absence of finding f_j ?"
7. Is it possible that the hypotheses in the diagnostic conclusion do not really explain the findings? For example, "What is the connection between hypothesis h_i and finding f_j ?" and "Isn't finding f_j far too extreme to be completely explained by hypothesis h_i ?"
8. Might the hypotheses in the diagnostic conclusion be rated too high? Included here are such questions as, "How does the data support hypothesis h_i ?" and, "Isn't hypothesis h_i inconsistent with finding f_j ?"
9. Has some hypothesis been underrated? This group includes, "Isn't hypothesis h_i often found in cases where hypothesis h_j is present?" and, "What makes you think the data does not support h_i more strongly?"¹

Furthermore, these questions express the only reasonable concerns that arise *solely because* it is a diagnosis system. Other appropriate questions, not about diagnosis as such, might include requests for definition of terms, exam-like questions that check the system's knowledge, and questions about implications for treatment. We are not

¹ Throughout this section, and the following two sections, "implausible" is used to mean "ruled out," "plausible" means "not ruled out," and "rating" means the final rating of non-ruled-out hypotheses.

suggesting that all questions will be in exactly one of these classes, some questions may refer to many of these concerns.

Consider an example of liver disease diagnosis given by Harvey and Bordley [19]. In this case the physician organizes the diagnosis around hepatomegaly (i.e., enlarged liver) giving 5 types of causes of hepatomegaly: venous congestion of the liver, obstruction of the common duct, infection of the liver, diffuse hepatomegaly without infection, and neoplasm (i.e., tumor) of the liver. He then proceeds to describe the evidence for and against each category. Venous congestion of the liver is ruled out because none of its important symptoms are present. Obstruction of the common duct is ruled out because it would not explain certain of the important findings and many expected symptoms are not present. Various liver infections are ruled out due to inability to explain certain important findings, though one of the infections seems fairly likely. Diffuse hepatomegaly without infection is considered implausible because, by itself, it is not sufficient to explain some important findings. Neoplasm is plausible and would adequately explain all the important findings. Finally he concludes:

The real choice here seems to lie between an infection of the liver and neoplasm of the liver. It seems to me that the course of the illness is compatible with a massive hepatoma [neoplasm of the liver] and that the hepatomegaly, coupled with the biochemical findings, including the moderate degree of jaundice, are best explained by this diagnosis. [19, p. 302]

In terms of the shared model of diagnosis:

- The principal complaint is hepatomegaly.
- The diagnostic hypotheses that might explain the principal complaint are: venous congestion of the liver, obstruction of the common duct, infection of the liver, diffuse hepatomegaly without infection, and neoplasm of the liver.
- Venous congestion of the liver, obstruction of the common duct, and diffuse hepatomegaly without infection are ruled out.
 - * Obstruction of the common duct and diffuse hepatomegaly without infection do not actually explain the principal complaint.
 - * Venous congestion of the liver is implausible.
- The diagnostic conclusion is neoplasm of the liver since it is the best of the plausible explanations that are left (infection of the liver, neoplasm of the liver).

Interestingly, in this case Bordley's diagnosis is wrong. Autopsy revealed that the patient actually had cancer of the pancreas.² Based on the above analysis, the

²To be fair, the autopsy also found tumors in the liver, but pancreatic cancer was considered the primary illness.

physician could have missed the diagnosis only for one or more of the following reasons:

1. Hepatomegaly may not actually be present.
2. There may be other important findings that should count as part of the principal complaint.
3. There might be causes of hepatomegaly that are not known to the physician or were overlooked by him.
4. Maybe venous congestion is more plausible than he thinks it is.
5. Obstruction and diffuse hepatomegaly might explain findings that the physician thinks they cannot explain.
6. Neoplasm might actually be implausible.
7. Neoplasm might not actually explain the principal complaint.
8. Perhaps neoplasm should be rated lower than it is.
9. Perhaps infection should be rated higher than it is.

Continuing the example, if the physician were asked, "What makes venous congestion implausible?" he might answer,

This patient exhibited no evidence of circulatory congestion or obstruction of the hepatic veins or vena cava [19, p. 301]

thus trying to convince the questioner that venous congestion was correctly ruled out. If asked, "Why not consider some toxic hepatic injury?" the physician could reply,

[It would not] seem to compete with a large hepatoma in explaining the massive hepatomegaly, the hypoglycemia, and the manifestations suggestive of infection. [19, p. 302]

thus trying to convince the questioner that the differential is broad enough. If we asked the physician, "How do you account for the sharply elevated amylase?" his only possible reply would be,

Oops.

because the amylase is a significant finding in the case that is not explained by the diagnostic conclusion. The physician missed pancreatic cancer because one of the significant findings in the case, elevated amylase, was not included in the principal complaint (see 2 above).

2.2 Similarity of Other Approaches to Diagnosis

There has been a considerable amount of work on diagnosis in medicine and in AI. In this section I will describe several approaches to diagnosis and show that they all contain, at least implicitly, the shared model of diagnosis presented in the last section.

Medicine has a long history of conceptual and theoretical study of diagnosis. The most common way that diagnosis is taught and described is called *differential diagnosis* [18]. *Stedman's Medical Dictionary* defines differential diagnosis as:

The determination of which of two or more diseases with similar symptoms is the one from which the patient is suffering. [20, p. 389]

As presented in medical textbooks differential diagnosis is almost identical to the shared model of diagnosis. Based on the data available about the patient—from the patient's presenting complaint, history, physical examination, and laboratory tests—the physician identifies the relevant problem area, i.e., the set of symptoms that appears to be the key to the patient's illness. Then he considers diseases that might cause those symptoms. This set of diseases is called the differential. Next he examines the differential attempting to decide which diseases can be ruled out on the basis of the evidence gathered so far, or on evidence that might be gathered, and to decide which of the diseases is most likely. In the end, the diagnosis is the disease, or set of diseases, that the physician considers to be most likely of those that are not ruled out. So the principal complaint is the relevant problem area. If we equate "causes" with "explains" then the diagnostic hypotheses are the diseases in the differential. Some of them are ruled out because, in the given case, they do not actually explain the symptoms. Some are ruled out because they predict certain findings that are not present or the pattern of findings is not consistent with the presence of the disease. Finally, the best hypotheses that are left are chosen as the diagnosis.

One of the important tasks of MYCIN is identifying infecting organisms [2]. MYCIN does not explicitly represent the problem in abduction terms, nor have its authors ever described it in those terms. However, MYCIN did have an implicit model of diagnosis as was made clear by Clancey's re-examination of MYCIN's rules for the purpose of building a tutoring system. This led to the development of NEOMYCIN [13], which performs MYCIN's diagnosis task with a more explicit representation of the problem-solving strategy. NEOMYCIN begins with a "chief complaint," which is used to generate some hypotheses, based on the diseases known to cause it. The process of establishing these hypotheses generates additional data that may in turn suggest new hypotheses. Eventually the hypotheses are fully refined and scored, no new hypotheses are suggested, and the program terminates because there is nothing more to do. The suggested hypotheses are those that cause pertinent findings. The domain rules score hypotheses based on several factors including what they explain and the presence or absence of manifestations normally associated with the disease. In the end MYCIN/NEOMYCIN produces an ordered list of organisms that might be causing the infection and that have a score above a certain threshold. So the principal

complaint is the "chief complaint" together with other findings that may turn up and need explanations. The diagnostic hypotheses are given in a hierarchical space of hypotheses. Some hypotheses are ruled out because they do not explain anything (implicitly by the initial selection process, equating "causes" with "explains"). Other hypotheses are ruled out by the evaluation process since hypotheses with a value below a plausibility threshold are removed from further consideration. A set of the highest rated organisms, i.e., the best of the remaining hypotheses, are presented in order of their values as the diagnosis.

At the opposite extreme from MYCIN with its *implicit* model of diagnosis is Reiter's explicit theory of diagnosis from "first principles" [30]. In Reiter's theory diagnosis requires a set of components and a description, in any deductive logic, of the normal behavior of the device being diagnosed. The diagnostic problem is set by an observation that conflicts with expected behavior. The problem is, "to determine those system components which, when assumed to be functioning abnormally, will explain the discrepancy between the observed and correct system behavior." [30, p. 58] A diagnostic conclusion is a minimal set of abnormally functioning components that is logically consistent with the observation. Thus the principal complaint is an observation that is inconsistent with the expected behavior. The set of diagnostic hypotheses is the power set of the set of all propositions, $ABNORMAL(e)$, for all components e . A number of these hypotheses can be ruled out since they are not consistent with the observation. Thus, all rule outs are on explanatory grounds since Reiter equates "explains" with "is consistent with." Reiter considers the problem to be one of generating *all* diagnoses, so the final step of choosing the best diagnosis is not done.

INTERNIST is a knowledge system that performs diagnosis in internal medicine [24] and explicitly represents the diagnosis problem in abductive terms by attempting to model differential diagnosis. Its disease representation includes with each disease a set of manifestations known to occur in association with the disease. It also associates two numbers with the link between each disease and each of its manifestations that qualitatively represent *evoking strength*, how strongly to believe the manifestation is caused by the disease, and *frequency*, how often patients with the disease have the manifestation. Also each manifestation has an *import* assigned, that is, how necessary it is that the diagnosis explain the manifestation. In addition, there are links between diseases and between manifestations that express causal and predisposing relationships. All diseases evoked by the observed manifestations are rated based on the evoking strengths, frequency, import, and their relations to other diseases that have been concluded or ruled out. Diseases that score below a threshold are discarded. If the top-ranked disease has competitors INTERNIST selects a questioning strategy designed to eliminate them and the disease hypotheses are rescored. (Two diseases are competitors if taken together they explain no more observed manifestations than either does individually.) If the top-rated disease has no competitors then it is concluded, i.e., it is part of the diagnostic conclusion, and all manifestations it explains are removed from consideration and the hypotheses rescored. This process repeats until all manifestations with high import are explained. So the principal complaint is the

set of manifestations with high import. The diagnostic hypotheses are the power set of known diseases. All diagnostic hypotheses except the conclusion are ruled out, though the rule out combines explanatory ability with other reasons. Since everything else is ruled out, the diagnostic conclusion does not need to be chosen from the best of the remainder. However, since at each step only the top-ranked hypotheses can be concluded, it can be said that it is the best way of explaining some of the unexplained manifestations.

Reggia's generalized set covering model (GSC) [29] is an explicitly abductive diagnosis system, like INTERNIST, that contains some formal aspects. In GSC a diagnostic problem has a set of diseases, a set of manifestations, a relation that maps diseases to the manifestations they can cause, and a set of observed manifestations. An explanation is defined to be a minimal set of diseases that, taken together, can cause all the observed manifestations. The diagnostic problem is to find all explanations for the observed manifestations. In addition, GSC has a qualitative probability associated with each "disease can cause manifestation" relation that can be used to rank the diagnoses. And, since some diseases always cause certain manifestations, this information can be used to categorically reject some diagnostic hypotheses. Thus, in terms of the shared model, the principal complaint is the set of observed manifestations. The diagnostic hypotheses are the power set of the set of diseases. Some hypotheses can be ruled out, either because they do not explain any observed manifestations or because manifestations they expect to find are not present (i.e., independent of what they can explain). GSC can rank-order the remaining hypotheses based on a qualitative version of Bayes' Rule, however, it does not pick the best since Reggia considers the problem to be generating all explanations.

There have been many more diagnostic systems in AI than can be described here. I have tried to select a representative sample for the purpose of describing various views of diagnosis. MYCIN is representative of all rule-based diagnosis programs. Reiter represents a large area of research on diagnosis from "first principles" and an important non-medical view of diagnosis. INTERNIST and GSC are well-known and influential medical systems. In addition, these descriptions leave out many important aspects of the systems. For example, MYCIN contains a therapy-selection component; INTERNIST, and much of the "first principles" work, is concerned with "sequential diagnosis," i.e., using a tentative diagnosis to guide the selection of tests that will narrow the diagnosis. Finally, the space of diagnostic hypotheses in some cases may seem incredibly large (e.g., for INTERNIST, which contains about 500 diseases, there are 2^{500} diagnostic hypotheses!), but this does not imply that the space is searched exhaustively. These systems use many techniques to reduce the search: only certain hypotheses are suggested by the data (MYCIN/NEOMYCIN, INTERNIST, GSC), hierarchical search (MYCIN/NEOMYCIN), sequential explanation of parts of the data (INTERNIST), etc. But my purpose was not to describe these systems completely, or even to describe how they work, but to elucidate their models of diagnosis as a task.

From this brief survey it appears that the shared model of diagnosis presented in the last section is a common view of the diagnosis task. The only significant

difference is that some approaches (Reiter and GSC) do not select *the* diagnosis. Instead they find all possible diagnoses and stop. However, if the goal of diagnosis is to find out what is true in order to decide what to do about it, selecting *the* diagnosis is very important. There may be situations in which it is difficult or impossible to choose, or when confidence in the choice is low, but making the choice seems to be a fundamental goal of the task.

The last point, that the shared model of diagnosis given here is a common view of the diagnostic task, deserves some elaboration. I described several diagnostic systems, each with its own approach to diagnosis, and showed that they all include the shared model of diagnosis. I do not mean to imply by this that *all* approaches to diagnosis will share this view. In fact, there is one common competing view—diagnosis as description, i.e., the goal of diagnosis is to describe the patient's state, not to find a cause for the symptoms. This view is embodied, broadly, in the LAIR system MDX2 [34]. I only intend to present a model of diagnosis, argue that it is a common one, and show the value of the model in developing explanation. The model represents the logical structure of the task and can be taken to define the task. If users and systems do not agree on a model of diagnosis, it will be difficult for diagnostic systems to explain their actions and conclusions. But, if users and systems agree to a model, then it can be used to develop explanation for diagnosis in the manner I describe. The details will change if the model changes, but the method, and the idea, of using a shared model to develop explanations remains.

3 A Generic-Task Architecture for Diagnosis

The shared model of diagnosis presented in Section 2 is independent of how diagnosis is actually done. That is, the model expresses those concerns that are appropriate to the diagnosis task regardless of the method used by any particular knowledge system in performing diagnosis. In other words, there is no necessary relationship between the concerns expressed by the shared model of diagnosis and the process by which a particular diagnostic system solves problems. But users of diagnosis systems have a right to expect those concerns to be satisfied by the diagnosis however it is done. As a result, users will ask questions that relate to the shared model. Thus, the model does suggest characteristics that a diagnosis system should have in order to be explainable. It should be able to develop a differential, decide what each diagnostic hypothesis can explain, rule out hypotheses on various grounds, and choose the best hypothesis.

This suggests a simple procedure for diagnosis. This system would be equipped with a set of hypotheses and would attempt to explain all the data in each case. It would do this by first ruling out those hypotheses that it could, then it would evaluate the remainder and choose the best. Of course, such a system probably would not work very well in practice. Much of the data does not need to be explained, e.g., a patient's age and sex. The process of ruling out and rating hypotheses may suggest new data and some of this new data may need to be explained, i.e., the diagnostic process is not necessarily a simple, linear one. Additionally, the number of diagnostic

hypotheses in even simple situations might be huge. A more workable strategy would be to consider parts of the hypotheses individually and have some way of putting the parts together into diagnostic hypotheses.

3.1 An Architecture for Diagnosis

The architecture given here is essentially based on generic tasks [6]; however, generic-task theory does not include everything needed in diagnosis so some extra elements are necessary. In summary, the architecture uses the Generic Tasks *hierarchical classification* [5], to generate plausible hypotheses, and *abductive assembly* [23], to put together a diagnostic conclusion. The hypotheses produced by the classifier are assumed to be partial diagnostic hypotheses in the sense that in typical cases the diagnostic conclusion developed by the assembler will contain more than one of them. This has been called the *multiple-fault* problem in AI, but since it is the typical case, the shared model of diagnosis and the architecture given here deal with it as typical, with the single-fault problem being a special case. Other generic tasks involved include *hypothesis matching* [22] for recognizing signs of hypotheses in the data and *knowledge-directed data abstraction and inference* [25] for retrieving data on each case. In addition we assume the existence of a module for determining the causal consequences of hypotheses, i.e., what they can explain. Each of these components is best thought of as an independent problem-solving *agent*. The architecture specifies the functional roles of each agent relative to the others and to the diagnosis task. I will describe each of the modules in detail in connection with the example system described in Section 5. But the two main components, the classifier and the assembler, will be briefly described here.

The use of the term "hypothesis" might occasionally be confusing. A diagnostic hypothesis, or partial hypothesis, is a hypothesis about a condition that may or may not be true of the underlying system (i.e., the system being diagnosed). Thus, when a hypothesis is concluded, the diagnosis system believes that the hypothesized condition is true of the underlying system. And when a hypothesis *explains*, or *can explain*, the principal complaint it means that the hypothesized condition, if present, could *cause* the principal complaint. For example, in the medical case discussed in Section 2 the disease hypotheses are hypotheses about the patient's condition. The hypothesis "neoplasm" is the hypothesis that the patient has a neoplasm. And when the physician concludes "neoplasm" he believes that the patient actually had a neoplasm. When he says that the hepatomegaly is best explained by neoplasm he means that he believes neoplasm is causing hepatomegaly.

The hypotheses are organized into a hierarchical classification structure such that more general hypotheses are near the top of the hierarchy and more specific ones near the bottom. This hypothesis space is explored top down using the establish-refine strategy—if one hypothesis is established, i.e., considered plausible, then it is refined by attempting to establish the hypotheses immediately below it. Additionally, if a hypothesis is ruled out, all hypotheses below it in the hierarchy can be considered ruled out as well. Thus entire portions of the hypothesis space may be eliminated

without exploring them. Eventually there will be no more hypotheses to refine, either because all plausible hypotheses are terminal in the hierarchy or because no hypotheses are plausible enough to refine. The hypotheses produced for consideration by the abductive assembler are those plausible hypotheses along this "explored front" of the hierarchy.

The abductive assembler puts together a composite diagnostic hypothesis out of the individual partial hypotheses generated by the hierarchical classification component. It does this by first examining the data to find the most important finding to be explained. It then polls the hypotheses to find those that offer to explain the finding and selects the most plausible hypothesis as the best explanation of the finding. If a choice cannot be made, it focuses a different finding to explain. If a choice is made, it then reduces the data to be explained by all the data the chosen hypothesis offers to explain and chooses a new finding to explain from what remains. The assembler repeats this process until all the data that needs to be explained is explained or all the data that remains to be explained cannot be explained (nothing offers to explain it or the assembler cannot choose between the hypotheses that offer to explain it).

Problem solving begins with the classifier, which uses hypothesis matchers for each hypothesis. The hypothesis matchers use the data base to access the case data. The classifier runs until no more hypotheses can be refined. The assembler uses the data base to determine the overall principal complaint and then uses the plausible hypotheses produced by the classifier to assemble the best explanation it can find for the principal complaint. In the process the assembler uses the consequence finder to determine which hypotheses explain a particular finding and what findings are explained by the composite as it is put together. The assembler returns the best explanation as the diagnostic conclusion.

3.2 Relationship Between the Architecture and the Model of the Task.

The overall principal complaint is determined by the knowledge-directed data abstraction and inference component, i.e., by the data base. It does this using its special knowledge about data in the domain. In addition, the assembler focuses on one finding at a time and has a way of choosing the most significant finding around which to organize a differential. This process is equivalent to choosing a principal complaint. Note that the data base cannot do this since it is not really a property of data that it can be used to make a good differential. The best way to make a differential depends mostly on the assembly strategy, so it is appropriate that it be done by the assembler. The assembly process can be viewed as choosing a principal complaint, explaining it and possibly other data as well, choosing another principal complaint from the data that remains, etc. Thus, the overall principal complaint, i.e., that part of the data that needs to be explained, is determined by the data base, and parts of the data can be recognized as local principal complaints by the assembler's choice of most significant finding for solving part of the problem.

The set of hypotheses that might potentially be used to explain the data is determined by the hypotheses in the classification hierarchy. That is, the builder of the system selected those hypotheses known to be useful for explaining data in the problem solver's domain (e.g., diseases of internal medicine for INTERNIST [24]). Additionally, in each case the consequence finder is used to decide whether or not a particular hypothesis can in fact explain some particular data. Finally, the true differential is the power set of the hypotheses in the classification hierarchy, since the diagnostic conclusion can contain many hypotheses. But the strategy of considering the hypotheses individually by the classifier and possibly ruling out some of them can reduce the number of composite hypotheses considerably. Then the strategy of assembling the composite one hypothesis at a time makes it unnecessary to explicitly consider even this differential. However, this strategy may result in overlooking possible conclusions that are as good as the one produced by the assembler since it produces a best explanation, not *all* best explanations.³ So part of the differential is determined by the system builder while building the hierarchical classification component. Part of the differential is determined by the consequence finder during problem solving by producing a list of hypothesis capable of explaining the finding the assembler is working on at each step. The differential is kept to a manageable size by the problem-solving strategy of classification to reduce the number of individual hypotheses and sequential assembly to reduce the number of composite hypotheses considered.

As already mentioned, the consequence finder is used to decide what an individual hypothesis can explain when the assembler is developing the differential. It is also used by the assembler to decide what assembled composite hypotheses can explain. Some points about explanatory interactions should be mentioned here. Consider just pairwise interactions for now. There are three possible explanatory interactions between two hypotheses:

1. The two hypotheses together may explain exactly what they explain separately. This would be the normal case, particularly when they are independent, i.e., not causally or otherwise related, and do not offer to explain any of the same findings.
2. They may jointly explain more than they do individually. An example of this is when they both offer to explain the same finding, both are capable of explaining a higher than normal value, the net effect being that jointly they can explain a greater increase than either can separately.

³If the assembler is able to build a complete explanation, it will have the property that: (a) each individual hypothesis in it is a best explanation for some finding, and (b) removing any single hypothesis will make the explanation incomplete. In these terms there are no better explanations, though there may be many that are as good. If additional criteria are considered, such as preferring smaller numbers of individual hypotheses in the composite, then there may be composites that are better on these grounds than the assembler's answer (e.g., composites that are as good as the assembler's best explanation but have fewer parts). This is a difficult issue. Bylander, et al., [3] have considered several criteria for "best explanation," their various difficulties and advantages.

3. Or they may jointly explain less than they do separately. An example of this is when they both offer to explain the same finding, one tending to explain a higher than normal value and the other a lower than normal value, the net effect being that jointly they cannot explain any deviation. .

The last of these would make things difficult for the assembler since, if it were possible in the domain, there would be no guarantee of explanatory progress when hypotheses are added to the composite. If such things were common the assembly mechanism would not be appropriate. But if they are uncommon then the assembler generally would make progress. If hypotheses always explain jointly exactly what they explain separately then the job of finding out what a composite can explain is easy. But if there are interactions, determining the explanatory power of a composite becomes more difficult. This becomes especially so if it is necessary to consider interactions involving three, four, or more hypotheses. I am not offering a theory of how to do this in general, nor is there a general theory available anywhere else. In the absence of effective, general solutions, ad hoc means must be used.

The hierarchical classification component takes care of ruling out those hypotheses that are implausible independent of what they may be capable of explaining. This is done partly by the hypothesis matchers, which may explicitly rule out hypotheses, and partly by the classification strategy itself, which causes hypotheses below an explicitly ruled out hypothesis to be implicitly ruled out. Other hypotheses are implicitly ruled out on explanatory grounds by the assembler because they do not explain any of the significant findings examined during problem solving. That is, a hypothesis will not be considered if everything it could explain can also be explained by other hypotheses that are best explanations of more significant findings.

The assembler chooses the best hypothesis for each finding by ranking the hypotheses by plausibility and choosing the highest-ranked hypothesis. The composite hypothesis it forms is explicitly compared only to subsets of itself, the smallest subset that is complete is considered the best. Other composites are only implicitly considered by the assembly process—there might be other composites equally good in the sense that they are complete explanations that contain no complete explanations as proper subsets. However, the composite found by the assembler is better in the sense that each of the significant findings is explained by the best available hypothesis whenever possible.

Summarizing, the concerns raised by the shared model of diagnosis are addressed by the architecture given here as follows:

Recognizing the Principal Complaint: Overall the principal complaint is recognized by the data base, and at each step of its processing the assembler is choosing a principal complaint when it chooses the most significant finding.

Developing the Differential: This is done partly by the system builder in designing the classification hierarchy and partly by the assembler using the consequence finder at each step to explain the most significant unexplained finding.

Deciding What Can Be Explained: This is done by the consequence finder for both individual hypotheses and composites.

Ruling Out: This is done mostly by the hierarchical-classification component on plausibility grounds, partly by the assembler on explanatory grounds.

Choosing the Best: This is done by the assembler at each step based on plausibility values set by the hypothesis matchers, and done overall implicitly by the assembler's strategy.

It should be emphasized that the architecture given here is not the *only* architecture possible for diagnosis. It is also not the only possible architecture that is responsive to the concerns expressed in the shared model of diagnosis. Nor is it the only possible generic-task architecture. However, it is an architecture that has emerged after a fairly long period of generic task research into diagnostic problem solving. In addition it seems to be particularly appropriate to the shared model of diagnosis.

4 Explaining Diagnosis

In the last two sections I have presented a model of the diagnosis task and an architecture for performing diagnosis that is particularly appropriate for the task. The model makes evident a number of concerns that might be raised as questions by users of diagnosis systems. In giving an architecture for diagnosis, I also described how a system with that architecture meets each of the major concerns in the shared model of diagnosis. I did not, however, describe how such a system actually answers the diagnosis questions. That is what this section is about.

4.1 Outline of the Approach

One possible approach for developing explanation for a diagnosis system is suggested by the shared model of diagnosis. It is possible to derive from the model a list of question types that a diagnosis system must be able to answer (see Section 2). Furthermore, given a diagnostic architecture that identifies which parts of the system are responsible for each diagnostic goal (Section 3), we can identify the parts of a diagnostic system that will be responsible for answering each question. However, it is not possible to derive user-level questions from the model because the model is very general while user questions are very specific. So, to get a usable explanation facility, we need to find out what people actually want to ask. This is the approach used here:

1. Derive the diagnostic question types from the shared model of diagnosis.
2. Survey people to find out what questions they ask.
3. Generalize these questions slightly.
4. Classify the questions into the diagnostic question types, those that do not go in any category are not diagnostic questions
5. Decide how each question should be answered based on the architecture and the model.

In this section I will describe this process, beginning with a representative sample of questions that were actually asked of a diagnosis system, and ending with a description of how the answers to the questions can be found in specific parts of a system that was built using the diagnostic architecture given in Section 3.

4.2 Questions Related to the Model of Diagnosis

The questions given here all pertain to a particular medical diagnosis system called RED, a red-blood-cell antibody identification program. Briefly described, RED's input, the data to be explained, consists of a set of "reactions" that are presumed to be caused by antibodies in the patient's blood; RED's output is a set of antibodies that seem

most likely to be causing the reactions (its "best explanation") and a rating of its confidences in the antibodies (its "final classification"); RED's architecture is roughly like that described in Section 3 with a classification hierarchy of antibody hypotheses, a data base capable of making various useful inferences about the data, an abductive assembler, and a mechanism for determining what is explained: More details on RED's domain, and a version of the program, will be presented in the next section.

4.2.1 The Questions That Were Asked of RED

The questions given here were derived informally from several sources. In this section I will describe a representative sample of these questions and give answers to them. The answers are based on knowledge that RED has, either explicitly or implicitly. RED does not actually produce these answers, nor is it necessarily trivial to modify RED to do so. Note that in the following I use, e.g., "S" to refer to the antigen known as S and to the antibody to the S antigen. When the meaning might be unclear I use "S antigen" and "S antibody" respectively.

People often want to know about the ratings, or confidence values, of the hypotheses. Questions such as,

Why did N get an initial confidence value of 1?

which would be answered,

There are reactions that N could explain but N is a rare antibody..

The hypothesis matcher simply describes the features of the case that matched and how they matched the expectations of the antibody. Another question about plausibilities is,

Why did Fy^a get ruled out?

which would be answered,

Fy^a was ruled out because a cell with strong expression of the antigen, 506A, failed to react.

This is the general principle that an antibody will react where it gets its best chance. If there is no reaction there, the antibody must not be present.

Another body of questions has to do with the diagnostic conclusion, or best explanation. People want to know why hypotheses are in it and why others are not. For example,

Why is S included in the answer?

which would be answered,

Because S is the best way to explain the reaction in Coombs on cell 623A.

This answer prompts questions like,

Why is S the best explanation for the reaction in Coombs on cell 623A?
which is answered,

S is the most plausible of the alternative explanations for the reaction.

In other words, the system has asserted that S is in the best explanation because it is the best explanation for that reaction, the question asks why. The response is that it is the most plausible way of explaining the reaction.

People ask many questions that cannot be answered by RED for a variety of reasons. These include.

Have you ever seen a case like this before?

where they want to know that similar cases have happened before, with similar solutions. Since RED has no memory of its past cases, and such memory is not part of the diagnostic architecture given here, it cannot answer this question. More importantly, this question is one that could be asked of any problem solver and is not specific to diagnosis. People also ask questions that are completely outside RED's domain, e.g.,

Why did you choose this unit of blood?

RED's output might be used to decide which unit of blood to choose, but RED does not choose blood, it merely identifies antibodies. People ask many more questions than I have described in this section. The representative sample given here will serve to illustrate the question development process and the method of relating the questions to the diagnostic architecture and model.

4.2.2 Generalization of the Questions

So far the questions have been given as specific questions about specific entities in RED. In this section I present generalized versions of the questions. The aim is to derive questions that apply to any diagnosis system.

The question, "Why did Fy^a get ruled out?" easily generalizes to,

Why was h ruled out?

where h represents any diagnostic hypothesis. Another question about the initial rating of the hypothesis, "Why did N get a confidence value of 1?" generalizes to,

Why does h have rating r ?

where r is a rating value assigned to hypothesis h by the system. In other words, Why does the system consider h to be plausible to a certain degree? The answer should be a description of the reasons for assigning rating value r to hypothesis h .

In general a hypothesis gets a certain rating because the data in the case match the expectations for the hypothesis to a certain degree. This question is asking about those expectations and how they are met.

The questions about why hypotheses are in the best explanation can be generalized as:

Why is h in the best explanation?

This includes, "Why is S included in the answer?"

The question, "Why is S the best explanation for the reaction in Coombs on cell 623A?" can be generalized to:

Why is h the best way to explain f ?

Where f is a finding and h is the hypotheses the assembler chose to explain it. So the question means, "Why was h chosen to explain f ?"

4.2.3 Mapping the Questions to the Diagnosis Model

Recall that the shared model of diagnosis gives rise to several types of questions. In this section I will relate the questions given in the previous section to those question categories suggested by the model of diagnosis. (See Section 2.1 for the list of diagnosis question classes.)

The question, "Why was h ruled out?" relates directly to the question class:

Has some hypothesis been incorrectly ruled out?

And the question, "Why is h in the best explanation?" belongs here also since one reason why h is in the best explanation is that h is not ruled out, i.e., h is available to explain the data (perhaps it should not be).

The question, "Why is h in the best explanation?" is in the question class:

Is it possible that the hypotheses in the diagnostic conclusion do not really explain the findings?

since part of the reason why h is in the best explanation is that it offers to explain one or more of the important findings. This might be an error.

If a hypothesis is in the diagnostic conclusion, the reasons why it has the rating it has can be used to determine whether or not it is overrated. So the question, "Why does h have rating r ?" goes in the question class:

Might the hypotheses in the diagnostic conclusion be rated too high?

The question, "Why is h in the best explanation?" also belongs here because one of the reasons why h is in the diagnostic conclusion is that it is the best explanation for something, possibly due to its rating. Another question that belongs in this class is, "Why is h the best way to explain f ?" since one reason why h is the best explanation for f may be that h is incorrectly rated higher than any of the alternative ways of explaining f .

The question, "Why does h have rating r ?" is also related to the question class:

Has some hypothesis been underrated?

because. If h is not in the diagnostic conclusion and not ruled out, the reasons why it has the rating it has can be used to determine whether or not it is underrated.

Note that some questions are in more than one of the question classes defined by the diagnostic model. The reason is that their answers have several parts, each part relating to one of the diagnostic concerns.

4.3 Questions Mapped to the Diagnosis Engine

In the last section I mapped the questions that have to do with diagnosis to the classes of questions derived from the model of diagnosis. In this section I will use that mapping to relate those questions to the diagnosis architecture given in Section 3 to show which parts of the diagnostic problem solver are capable of providing answers, or parts of answers, to each question.

1. Why was h ruled out?

There are two ways to rule out a hypothesis, each with its own explanation:

- (a) If h was ruled out directly, the decision was made by h 's hypothesis matcher in the classification hierarchy, so the answer in this case comes from the hypothesis matcher.
- (b) Otherwise h was ruled out implicitly by the classification strategy because an ancestor of h in the classification hierarchy was ruled out. In this case the answer comes from the classifier and the ancestor's matcher.

2. Why is h in the best explanation?

The assembler picked h as best for some particular finding and will know which finding it was best for and why it was best. So the answer comes from the part of the assembler that chooses hypotheses to explain findings.

3. Why does h have rating r ?

Hypotheses are rated by the hypothesis matchers, so h 's matcher is responsible for answering this question.

4. Why is h the best way to explain f ?

This is answered by the part of the assembler that chose h to explain f . That is, the assembler focused on f , and decided, according to its criteria, that h was the best of the alternative ways of explaining f . Thus, this "chooser" is responsible for answering the question in this case.

In this section I began with questions that a person might want to ask of a particular diagnosis system, proceeded to generalize these into questions about diagnosis systems in general, and grouped those into the question classes identified in Section 2. Given that the questions were identified with diagnosis concerns, and that

the diagnosis concerns were identified with parts of a system architecture, it was then possible to identify the parts of the system architecture where the information needed to answer the questions is encoded. In other words, given a model and an architecture for the task it is possible to identify the questions the problems solver should be able to answer and say how to answer them.

5 Example Implementation—RED

RED is a medical diagnosis system that operates in the domain of hospital blood banks. The blood bank is a medical laboratory responsible for providing safe blood for transfusion. One of the major activities required to do this is red-cell antibody identification, which is the activity RED performs.

5.1 Problem Domain

In the blood-bank context, there is a patient who needs blood, the blood bank has an inventory of blood and must select a unit to give the patient. Blood cells have chemical structures on their surfaces called antigens that, when a donor's cells are transfused into a patient, can be recognized as foreign by the patient's immune system. When this happens the immune system produces antibodies that circulate in the blood serum and react with the donor's blood cells. The medical consequences of such transfusion reactions may range from fever and anemia to life-threatening coagulation disorders and kidney failure. Therefore, the blood bank must take care to ensure that the patient does not have antibodies to the donor's blood.

The first step in the testing process is determining the ABO group. Then, the patient's serum is screened for antibodies by testing it with two cells⁴ that have all of the clinically significant red-cell antigens. The test involves mixing the patient's serum (where the antibodies would be if there were any) with the screening cells to check for reactions. If reactions occur, then the patient does have atypical antibodies, which now must be identified.

The red-cell antibodies are identified by using a red-cell panel. This is a selection of cells that are typed for as many of the common antigens as practical. This information is recorded in a table called an "antigram." An example of an antigram is shown in Figure 1. In the figure the cells are named along the left side (164, etc.) and the antigens along the top (C, D, etc.). In the body of the table 0 indicates absence of an antigen and + indicates presence. For example, cell 209A has C but not D. The patient's serum is mixed with each of these cells, under several test conditions, and the reactions are noted on the test panel. Remember that the antibodies, if any, are in the serum, so a reaction on the test panel is caused by an antibody in the patient's serum. Figure 2 shows part of a test panel from RED. In the figure the various test

⁴"Cell" in the blood-bank domain typically refers to a small sample of cells from the same donor. So the two screening cells referred to here are not just two cells but two groups of cells, each from a single donor.

	C	D	E	c	e	C ^w	V	K	k	Kp ^a
164	0	0	0	+	+	0	0	0	+	0
195	0	0	0	+	+	0	0	+	+	+
186A	0	0	0	+	+	0	0	0	+	0
209A	+	0	0	+	+	0	0	+	+	0
303A	0	0	+	+	+	0	0	0	+	0
506A	+	+	0	0	+	+	0	0	+	0
537A	+	+	0	0	+	0	0	+	+	0
479	0	+	0	+	+	0	+	0	+	0
623A	0	+	+	+	0	0	0	0	+	0

Figure 1: Part of an Antigam Panel from RED

	623A	479	537A	506A	303A	209A
Albumin IS	0	0	0	0	0	0
Albumin 37	0	0	0	0	0	0
Coombs	3+	0	3+	0	3+	3+
Enzyme IS	0	0	0	0	0	0
Enzyme 37	0	0	1+	0	0	+

Figure 2: Part of a Test Panel from RED

conditions, or phases, are listed along the left side (Albumin IS, etc.) and the names of the cells are given across the top (623A, etc.). The panel matrix shows the reactions graded from 0 for no reaction to 4+ for the strongest reaction. So, e.g., 623A has a 3+ reaction in the Coombs phase. By examining the pattern of reactions on the test panel, and correlating it with the antigen information given in the antigam, the blood-bank technologist can identify the antibodies in the patient's serum. The technologist is looking for such features as the relative strength of reactions and whether cells with a particular antigen consistently react or fail to react. (This brief overview is based mostly on the paper by Smith, et al., [33].)

The antibody identification problem is complex and very difficult for people to solve. It is not really diagnostic, and most physicians do not regard it as such. What makes it interesting, and relevant here, is its abductive nature. The stated model of diagnosis presented in Section 2 is actually a model of abduction. Diagnosis is an abduction problem with the additional assumption that in diagnosis the explanatory hypotheses are assumed to represent diseases, or malfunctions. Although in this section I present an implementation of explanation in RED, which solves an abduction problem, the results should be applicable to any diagnosis system.

5.2 System Architecture and Implementation Details

Several versions of RED have been built, though only one (locally known as RED2) has been described in detail in published papers [23, 33]. The version I will describe here is very similar to RED2. Rather than deal with version numbers, though, I will simply call it RED. The system architecture of RED is very similar to the diagnosis architecture described previously. Two major differences are that there is no consequence finder as a separate module, and the overall principal complaint is given as part of the input to the system.

Data Base The data base contains information on a specific case including the patient's known antibodies and antigens, and the test panel with its associated antigram. RED is designed to explain the data in the test panel, i.e., the overall principal complaint is the set of positive reactions in the test panel.

Classifier The hierarchical classification component of RED is not really a hierarchy, having only one important level, and serves mainly to evaluate the hypotheses. It is probably best to think of this component as a set of antibody specialists, i.e., agents or modules, each of which performs two tasks:

1. Creates a "profile" of reactions it can consistently explain. Thus performing the job of the consequence finder.
2. Runs a hypothesis matcher to rate, or rule out, the hypothesis.

To create a profile the hypothesis builds a subset of the test panel that contains only the cells that are positive for the corresponding antigen. For example, the C specialist's profile would contain only cells 209A, 506A, and 537A based on the antigram in Figure 1. Then it adjusts the reactions to make them consistent with the behavior of the antibody. In the case represented by the test panel in Figure 2 and the antigram in Figure 1, C is strong on cell 537A and weak on cells 209A and 506A. So the reactions that C can explain will be weaker on 209A and 506A than on 537A. Additionally, there cannot be much variability in the reactions among the weak cells (or among the strong cells), so the reactions that C can explain on 209A will be much reduced from the reactions actually observed there since 506A did not react at all. After all the adjustments are made to the profile it will look like Figure 3. Compare this with the reactions observed on these cells in Figure 2.

Hypothesis Matcher After determining what it could potentially explain, each antibody specialist calls a hypothesis matcher to rate its plausibility. In fact, all specialists call the same hypothesis matcher, since the logic is the same for each antibody. (RED2 used a more sophisticated set of matchers.) Plausibility is rated on an integer scale from -3 (lowest rating) to +3 (highest rating). The first thing the matcher does is attempt to rule out. If there are no cells with the antigen, or if the profile contains only zeros, or if a strong cell failed to react, or if any cell totally failed to react,

	537A	506A	209A
Albumin IS	0	0	0
Albumin 37	0	0	0
Coombs	2+	0	+/-
Enzyme IS	0	0	0
Enzyme 37	1+	0	+/-

Figure 3: Profile for C

then the antibody should be ruled out. The matcher also checks to see if there is history evidence to use in rating the hypothesis. Patients known to possess the antigen cannot have the antibody⁸, whereas patients known to possess the antibody probably still have it. These two factors are combined with the known overall prevalence of the antibody to produce the plausibility rating of the hypothesis. Any hypothesis rated -2 or -3 is considered ruled out and will not be made available to the abductive assembler.

Assembler The final component of RED is the abductive assembler. In overview the process is:

1. Attempt to assemble a complete explanation by making only the "easy" decisions, i.e., choose a hypothesis if there is a clear reason to include it.
2. Attempt to explain the remaining data by making "hard" decisions, i.e., choose a hypothesis if there is any reason to include it.
3. If the result is not a complete explanation, make a judgment about whether or not a complete explanation might be possible.

The assembly procedure, steps 1 and 2 above, is: for each unexplained finding, until a complete explanation is found,

1. Collect the hypotheses that are capable of explaining the finding.
2. Choose the "best" hypothesis.
3. Add the chosen hypothesis to the composite hypothesis being built.
4. Reduce the unexplained findings by the findings that the new hypothesis explains.

The definition of "best" depends on whether it is making "easy" or "hard" decisions.

⁸ People can form antibodies to their own antigens, known as autoantibodies. RED is explicitly concerned only with antibodies to non-self antigens, i.e., alloantibodies.

Category	Hypotheses
Best Explanation	S, K
Confirmed	S
Likely Present	K
Likely Absent	Kp ^a , Js ^a , Lu ^a , Le ^b P ₁
Ruled Out	D, C, c, E, e, C ^w , M, N, s, Fy ^a , Fy ^b , Jk ^a , Jk ^b , k, Kp ^b , Js ^b , Lu ^b
Unresolved	Le ^a
Undetected	f, I

Figure 4: RED's Final Report

The Final Report After the assembler finishes, RED produces a "final report" as shown in Figure 4. In this report the antibodies are all put into categories that represent the degree to which RED believes that they are present. The categories are given on the left and the antibodies on the right, with lines connecting each antibody to its category. For example, Le^b is considered Likely Absent. The first category listed is the best explanation, the rest are degrees of confidence based on whether or not the antibody is in the best explanation and how high its rating is. The categories are defined as follows:

Confirmed: The hypothesis is essential.

Likely Present: The hypothesis is in the best explanation, plausible, but not essential.

Likely Absent: The hypothesis is not in the best explanation and not highly rated.

Ruled Out: The hypothesis is ruled out by the hypothesis matcher and tests were performed that would have shown evidence for the hypothesis if it were true.

Unresolved: The hypothesis is in the best explanation but it is not essential and not plausible, or it is not in the best explanation but it is highly rated. This category is intended to catch hypotheses with some conflict in their results, either the assembler was forced to use them even though they are not very good or they looked pretty good but the assembler did not use them.

Undetected: The hypothesis explains nothing but was not ruled out or it was not tested. This category catches conflicts as well, but of a different sort. These are hypotheses that either offer to explain nothing even though there is other evidence to make them plausible or the tests to turn them up were not done.

In RED's domain only certainty, i.e., Ruled Out and Confirmed, counts. So, in the result shown in Figure 4, more testing is needed, though it could be focused on those antibodies that were not ruled out or confirmed.

5.3 Details of Explanation Features

The final report described above, as displayed by RED, is an active object. Selecting any of the objects displayed there brings up a menu containing the item "Explain". If Explain is selected, RED displays a menu of questions that it can answer. The questions are those that are appropriate to the object selected. For example, the only question appropriate to the category Likely Present is, "What is the definition of Likely Present?", whereas for a hypothesis there are many questions (e.g., see Figure 5). In this section I will describe how RED answers these questions.

As RED runs, it records what it does with each hypothesis along the way. So at the end RED has recorded for each hypothesis information about whether or not it is in the best explanation, whether it is essential, whether it was ever in the best explanation, and so forth. For the most part, this information is needed to solve problems, but it is also used to answer questions about the hypotheses.

RED's explanation component has a list of questions appropriate to hypotheses, another list appropriate to ask of the overall diagnosis system, and a third list for the final report categories. The hypothesis questions are further annotated according to whether they are appropriate for any hypothesis, or only those hypotheses in the best explanation, or only those hypotheses not in the best explanation. These questions are all stored in template form. When something is selected in the final report, the explainer makes a question menu by choosing question templates from the appropriate list and filling them in. It also associates with each question a procedure for answering it, so when a question is selected from the question menu, the right procedure can be invoked. For example, selecting K in the final report (Figure 4) causes the explainer to select question templates for a hypothesis not in the best explanation. These are translated into the questions shown in Figure 5, which are displayed as the question menu.

5.4 Explanation Examples

In this section I will assume RED has finished working through one problem and displayed the final report in Figure 4. The relevant data for this problem is shown in Figures 1 and 2. I will give examples of questions RED can answer about the problem, and work through the process of answering these questions in detail.

Suppose the user selects K in the final report and asks for an explanation. RED would display a menu, through the process described earlier, containing the questions shown in Figure 5. These questions are the questions that can be asked of any hypothesis together with the questions that can be asked of a hypothesis in the best explanation (except for the questions whose answers are obvious from the final report

What does K explain?
 Why does K have plausibility 3?
 Why does K offer to explain a particular finding?
 Why doesn't K offer to explain a particular finding?
 Why is K considered Likely Present?
 What is K needed to explain?
 What are the alternatives to K?
 Why does K explain a particular finding?
 Why is K in the best explanation?

Figure 5: Questions Instantiated for a Hypothesis

itself). Suppose the user selects, "Why is K in the best explanation?" The explainer answers the question based on the template:

h is in the best explanation because it is (clearly the best/the best) way to explain:

f

since *reason*.

The phrase "clearly the best" is used if it was chosen by EasyDecisions and "the best" is used if it was chosen by HardDecisions. The finding listed is the finding the assembler had focused on at the time K was chosen. *Reason* is a text string that describes the reason K was chosen. So the explainer checks to find which decision maker chose K, what finding it was chosen for, and the reason it was chosen. The assembler records this information for each hypothesis. For K it will find the following:

chosen by: Easy
 chosen for: (537A Enzyme37 +/-)
 reason: StandOut

Then to translate the keyword "Easy" into English text the explainer looks on the table for the keywords the assembler uses to describe the procedures that choose hypotheses and finds:

Easy: clearly the best
 Hard: the best

To translate "StandOut" into text it looks on another table of keywords used to describe the reasons for choosing hypotheses and finds:

Essential: there is no way to completely explain the finding without it
 Only: it is the only hypothesis offering to explain the finding
 StandOut: it is both plausible on its own and much more plausible than its nearest competitor
 MostPlaus: it is more plausible than any of its competitors
 Suggested: it is suggested by hypotheses in the best explanation while none of its competitors are

From this the explainer gets all it needs to answer the question:

K is in the best explanation because it is clearly the best way to explain:

(537A Enzyme37 +/-)

since it is both plausible on its own and much more plausible than its nearest competitor.

The point here is not RED's ability to produce natural language text, it is not very good at it. Rather, my aim is to show that RED records the kind of information about its problem solving needed to answer the diagnosis questions. A good natural language generator could proceed from the keywords (e.g., Easy and StandOut) and other knowledge about the domain and context to produce much nicer, more flexible and appropriate, text. But the content of the answer is in the information recorded by the assembler during problem solving.

Suppose a user wants to know, "Why does K have plausibility 3?" The answer to this question is based on the template:

h is considered *SummaryKG* because the data *PresenceKG* the presence of *h* and *h* is a *PrevalenceVal* antibody. The data *PresenceKG* the presence of *h* because there are *RuleOutKG* to rule out and the patient *HistoryKG* the antibody. There are *RuleOutKG* to rule out because *ProfileEmptyVal* and *ProfileZeroVal* and *SignificantNonReactionVal* and *NonReactingCellVal*. The patient *HistoryKG* the antibody because *PossessAntigenVal* and *PossessAntibodyVal*.

The *xKG* items are the values of knowledge groups interpreted in English. A knowledge group is a small set of knowledge for making an abstraction of data that is significant for deciding the plausibility of a hypothesis. The hypothesis matcher in RED is implemented in terms of knowledge groups called Summary, Presence, and RuleOut. Similarly, the *yVal* items are the values of data base queries in the knowledge groups interpreted in English. If a knowledge group or data base query is not run, because the specialist could decide without it, the relevant clause is not included in the explanation. When K's hypothesis matcher ran, RED recorded the values of all the knowledge groups, and with each knowledge group it recorded the input values to the group. Looking up this information for K, the explainer finds:

SummaryKG:	3
PresenceKG:	2
PrevalenceVal:	3
RuleOutKG:	-2
HistoryKG:	0
ProfileEmptyVal:	F
ProfileZeroVal:	F
SignificantNonReactionVal:	F
NonReactingCellVal:	F
PossessAntigenVal:	U
PossessAntibodyVal:	U

Each of these values is translated into a phrase in a manner similar to the way described earlier for answering, "Why is K in the best explanation?" For example, looking up PossessAntibodyVal, the explainer finds:

- T: the patient is known to possess the antibody
- F: the patient is known not to possess the antibody
- U: it is unknown whether the patient possesses the antibody

All of these values are put together in the template to give the following answer for K:

K is considered very plausible because the data supports the presence of K and K is a common antibody. The data supports the presence of K because there are strong reasons not to rule out and the patient has no recorded history of the antibody. There are strong reasons not to rule out because there are cells with the antigen and there are reactions on at least some of these cells and no cell with strong expression of the antigen failed to react and all cells reacted. The patient has no recorded history of the antibody because it is unknown whether the patient possesses the antigen and it is unknown whether the patient possesses the antibody.

This answer is long and convoluted, but it demonstrates that the explainer can extract the information needed for the content of the explanation. How to present it, the exact form and amount of information to present, is a separate issue that I am not addressing here. This particular technique, having tables to translate values of all data base queries and knowledge groups into English phrases, only works in this version of RED because all the hypothesis matchers are the same. If they were all different this would get unwieldy very soon. The explainer should have some understanding of the domain and problem solving that is independent of RED itself so that it could properly interpret these values without an explicit translation being given.

There are many other questions RED can answer but these are a sample of the interesting ones. The detailed descriptions here should provide enough information to see how the rest are answered. All the questions are described in my dissertation [36] with a sketch of how RED derives the answers.

5.5 Implications for Designing Generic-Task Tools with Explanation Capabilities

Adding explanation to RED required determining the questions it needed to answer, based on the shared model of diagnosis, and where in the problem solver the answers would come from, based on the diagnostic architecture. Only the hierarchical classification and hypothesis matching components of RED were implemented using a generic-task tool—CSRL [5]—the rest of the system is implemented directly in LOOPS and Lisp. However, all the parts of the system are generic tasks for which tools exist or are in various stages of development: PEIRCE for abductive assembly [28], IDABLE for the data base [34], HYPER for hypothesis matching [22], and CSRL for hierarchical classification. The explanation component for RED has some implications for how these tools can include the right primitives for explanation.

There are three lessons for generic-task tool building. First, generic-task systems should be able to record the decisions they make and the facts pertinent to explaining each decision. Second, each generic task should have its own explainer, which uses its own terms, e.g., classifiers will explain in terms of classification goals, matchers in terms of match goals, etc. And third, in building problem-solving systems from generic-task components, we must relate the generic-task explanation terms with the problem-level terms so that explanations derived from the generic-task problem solvers can be sensibly presented at the problem level. Additionally, there still remains the job of getting domain terms into the explanations. I do not have a principled way of doing this. I suspect it is mostly a user modeling and user interface issue, which is not to say that it is simple to solve, but the problems are more likely to be problems about communicating with people than they are problems about knowledge organization and use for effective problem solving.

6 Extensions

A shared model of problem-solving tasks ought to have application to other tasks besides diagnosis and for other purposes than explanation. In this section I consider two of these extensions.

6.1 Application to Design

The main arguments for a shared model of problem-solving tasks are independent of the actual task itself. That is, knowledge systems are intended to solve problems of a particular kind, to perform a particular task; users understand this and expect that a system's actions and conclusions will be consistent with the goals of the task. So for each problem-solving task for which knowledge systems are built it should be possible to identify the shared model of the task and use it to guide the design of systems and help in the justification of their methods and conclusions.

The shared model of diagnosis given in Section 2 is approximately a logical definition of the diagnostic problem-solving task. It specifies the relationship between the input and the output that must hold for the task to be satisfactorily accomplished. There are other factors that are legitimately a part of diagnosis but are not so strongly part of it that they can be considered definitional. Consider that diagnosis is often performed not only to find out what is wrong but to decide what to do about it. That is, the symptoms represent an undesirable condition that one would like to repair. One strategy is to find out what is causing the symptoms and attempt to eliminate that. Thus, diagnosis is used to find the causes. So perhaps the shared model should include the pragmatic requirement that the conclusion, and the hypotheses from which it is selected, be something that can be treated, at least in principle. But sometimes nothing can be done and still diagnosis is appropriate and must meet the other criteria of the shared model. So along with the logical definition, or specification, of the problem-solving task a shared model may include pragmatic concerns that are not necessary to the task but are often part of it. In diagnosis these concerns seem to be of secondary importance. But that may not be the case in other tasks where pragmatic issues may be more important.

In order to develop a shared model for a task, the task must be very well understood. Diagnosis was a natural task to approach first because of OSU LAIR's long history of work on medical diagnosis and because of the considerable amount of published work on the topic in AI and in medicine. Another such task is design. There has been some interest in design within AI, and of course considerable interest in other fields, including engineering. We also have some experience with design in the LAIR [1].

In design, say, mechanical design, the goal is to design a device that will perform certain functions, subject to various constraints (or specifications) such as size, cost, materials, etc. Some constraints are absolute ("it cannot be more than a foot long") and some are not ("make it as small as possible"). But the implication for all the them is that the problem solver will attempt to meet all the constraints, and for those it fails to meet it will have good reasons. One important consideration in design is that the result be better than alternative designs, i.e., some sense that the design is the best that can be done. In addition, there are background assumptions that the device will be built and used. Sometimes this is explicit in the specifications and sometimes it is not. Of course, design might be done as a pure intellectual exercise so this is a pragmatic consideration. So, for a design system, users have a right to know that: (1) the designed device will achieve the desired functions, (2) the design meets as many of the specifications as possible, (3) there are good reasons for failing to meet the rest of the specifications, (4) there are no better designs, (5) various implicit, pragmatic, concerns are met. As with diagnosis, a design program could be implemented that did not do any of these things explicitly, but such a system may have trouble explaining its behavior or justifying its conclusions. If the program is built so that each of the design concerns can be identified with a part of the system, then it will be possible for the system to explain itself based on its own knowledge and a trace of its behavior. With some further work to expand on the model it could

be used to guide building a design program that can answer the questions it will get because it is a designer. The process would be analogous to that described in Sections 3 and 4 for diagnosis.

6.2 Learning

Suppose a diagnosis system, such as RED, is embedded in a larger system that has access to the "right" answer. If RED's answer is wrong then the shared model of diagnosis can be used to find out how it came up with the wrong answer and where new knowledge, or a change to existing knowledge, is needed. It will not help decide what the new knowledge is or how to change existing knowledge, but it can point out where the error occurred. If the containing system (i.e., the learner) had access to a source of knowledge (say, a human expert) it can ask what to do to fix its knowledge base. For example, the following trace may occur (my commentary is in brackets):

RED: The antibodies present are: K and Le^a.

Learner: The right answer is K and S. (to RED) Why isn't S in the best explanation?

[The Learner has two options here—find out why S isn't in or find out why Le^a is. Informed by knowledge of the assembly algorithm the learner first pursues S, since if S were put in then Le^a might have been removed by the parsimony critic.]

RED: S is not in the best explanation because it was not the best explanation for any finding it offers to explain since it was rated too low by its hypothesis matcher.

Learner: Why is S considered only somewhat plausible?

[Here, again, the Learner has two alternative lines of inquiry—find out why S was rated low or find out if S should be offering to explain some different findings. The reasons why S offers to explain the findings it does and not others may be complicated and getting all this may be time consuming. So the Learner pursues the rating.]

RED: Because the patient has no history of the S antibody and, even though there are reactions on all the cells with S antigen, S is a rare antibody.

Learner: (to a human expert) S is rated low because

Expert: Do you know that S reactions are enhanced by enzyme?

[The Learner simply passes the reasons along to a human expert who offers some possible new knowledge that RED apparently did not use in evaluating S.]

Learner: (to RED) Do you know ...?

RED: No.

[At this point, since the knowledge provided by the expert was not in RED, it would be added. RED should re-evaluate S, then check to see if the change is enough to get S into the best explanation and whether the new best explanation would be right.]

I do not imagine that doing all of this would be easy. But it is clear how the explanation from RED, together with the shared model of diagnosis, can be used to guide the search for the source of error. The QUAWS project in the LAIR is, in part, using ideas related to this for learning in medical diagnosis [21].

A system such as the one described above might even be able to start with no domain knowledge at all, only knowledge about the diagnostic task. When given case data it would begin asking questions of the expert based on the diagnosis model:

What data needs to be explained?

Is there any other data I should know about?

What hypotheses are likely to be relevant?

For each hypothesis:

How can the hypothesis be ruled out?

What should keep me from ruling it out?

Should it be ruled out in this case? Why?

What data is relevant to rating the hypothesis?

How should it be rated in this case? Why?

What can the hypothesis explain in general and in this case?

The system could then run, i.e., attempt a diagnosis. If it gets a wrong answer it could begin a dialog like the one above to find the source of error and ask for new knowledge. Such a system will be deficient in its knowledge organization and representation, but even so it should be interesting and may even be a way to show precisely how good representations (or their lack) affects the quality of problem solving.

7 Conclusion

The central idea of this paper is the shared model of a problem-solving task and its use in designing and explaining knowledge systems. It is called a "shared model" because it represents how both the knowledge system and the user understand the task. To illustrate this I presented a model of one particular problem-solving task—diagnosis. The model expresses a set of concerns that must be met by any diagnostic problem solver (i.e., the model defines the task; any problem solver that does not achieve all the goals of the model is inadequate as a diagnoser). Then I presented an

architecture for diagnosis in which each of the modules was identified with specific diagnostic concerns from the shared model of diagnosis. That is, the model tells what the concerns of diagnosis are and the architecture describes a problem solver that meets those concerns and, further, tells which parts of the problem solver meet which specific concerns. Additionally, a set of question classes can be derived from the model such that corresponding to each concern expressed by the model is a class of questions designed to assure the questioner that the concern is met. The next step was to derive the actual questions that a diagnostic system should answer. I began with questions that a person might want to ask of a particular diagnosis system, proceeded to generalize these into questions about diagnosis systems in general, then grouped those into the question classes derived from the shared model of diagnosis. Given that the questions were identified with diagnostic concerns, and that the diagnostic concerns were identified with parts of a system architecture, it was then possible to identify the parts of the system architecture where the information needed to answer the questions could be found. In other words, given a model and an architecture for the task it is possible to identify the questions the problem solver should be able to answer and describe how to answer them. Finally, I described a knowledge system called RED implemented using generic tasks, that makes use of these ideas to implement explanation.

One of the explicit assumptions I have made throughout this paper is that an explanation is based on inspecting the problem solver's structure and memory. But, in some cases these "introspective" explanations cannot be given. In particular this is true whenever: (1) the problem solver cannot be mapped to the model of diagnosis; or (2) the internal workings of the problem solver, or its memory for its actions, is not accessible; or (3) the problem solver's methods are not comprehensible or convincing. In all cases, the shared model of the task is still helpful in designing explanations and judging their adequacy. In fact, in the first case, the model says that the explanations must be based on something other than the problem solver itself.

Acknowledgments

This paper is based on the author's dissertation [36]. Thanks are due to my advisor, B. Chandrasekaran, and the rest of my reading committee: John Josephson, Terry Patten, and Jack Smith. The support given, in innumerable ways, by my wife DiAnne has been most valuable.

References

- [1] D. C. Brown and B. Chandrasekaran. *Design Problem Solving: Knowledge structures and control strategies*. Pitman, London, 1989.

- [2] B. G. Buchanan and E. H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA, 1984.
- [3] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. Some results concerning the computational complexity of abduction. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 44-54, Toronto, Ontario, Canada, May 15-18 1989.
- [4] T. Bylander and B. Chandrasekaran. Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, 26(2):231-243, 1987.
- [5] T. Bylander and S. Mittal. CSRL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7(3):66-77, August 1986.
- [6] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23-30, Fall 1986.
- [7] B. Chandrasekaran, J. Josephson, and A. Keuneke. Functional representations as a basis for generating explanations. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 726-731, Atlanta, GA, October 1986.
- [8] B. Chandrasekaran and S. Mittal. On deep versus compiled approaches to diagnostic problem-solving. *International Journal of Man-Machine Studies*, 19(5):425-436, November 1983.
- [9] B. Chandrasekaran, M. C. Tanner, and J. R. Josephson. Explaining control strategies in problem solving. *IEEE Expert*, 4(1):9-24, Spring 1989.
- [10] W. J. Clancey. Tutoring rules for quiding a case method dialogue. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 201-225. Academic Press, London, 1982.
- [11] W. J. Clancey. The epistemology of a rule-based expert system—a framework for explanation. *Artificial Intelligence*, 20(3):215-251, May 1983.
- [12] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289-350, December 1985.
- [13] W. J. Clancey and R. Letsinger. NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence*, pages 361-381. Addison-Wesley, Reading, MA, 1984.
- [14] R. Davis. Teiresias: Applications of meta-level knowledge. In R. Davis and D. B. Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 227-490. McGraw-Hill, New York, NY, 1982.

- [15] J. deKleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97-130, April 1987.
- [16] R. Duda, J. Gashnig, and P. Hart. Model design in the PROSPECTOR consultant system for mineral exploration. In B. L. Webber and N. J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 334-348. Tioga, Palo Alto, CA, 1981.
- [17] J. A. Goguen, J. L. Weiner, and C. Linde. Reasoning and natural explanation. *International Journal of Man-Machine Studies*, 19(6):521-554, December 1983.
- [18] A. M. Harvey and J. Bordley III. *Differential Diagnosis, the Interpretation of Clinical Evidence*. W. B. Saunders, Philadelphia, 1972.
- [19] A. M. Harvey and J. Bordley III. Illustrative case II in liver diseases. In *Differential Diagnosis, the Interpretation of Clinical Evidence*, pages 299-302. W. B. Saunders, Philadelphia, 1972.
- [20] W. R. Hensyl and J. O. Oldham, editors. *Stedman's Medical Dictionary*. Williams and Wilkins, Baltimore, MD, 24th edition, 1982.
- [21] D. E. Hirsch, S. R. Simon, T. Bylander, M. A. Weintraub, and P. Szolovits. Using causal reasoning in gait analysis. Technical report, The Ohio State University, Department of Computer and Information Science, Laboratory for Artificial Intelligence, Columbus, OH, 1988.
- [22] T. R. Johnson, J. W. Smith, Jr., and T. Bylander. HYPER—hypothesis matching using compiled knowledge. In *Proceedings of the Spring Symposium Series: Artificial Intelligence in Medicine*, pages 45-46, Stanford Univ., March 22-24 1988. American Association for Artificial Intelligence.
- [23] J. R. Josephson, B. Chandrasekaran, J. W. Smith, Jr., and M. C. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(3):445-454, May/June 1987.
- [24] R. A. Miller, H. E. Pople, Jr., and J. D. Myers. INTERNIST-1, an experimental computer-based diagnostic consultant for general internal medicine. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence*, chapter 8, pages 190-209. Addison-Wesley, Reading, MA, 1984.
- [25] S. Mittal, B. Chandrasekaran, and J. Sticklen. PATREC: A knowledge-directed data base for a diagnostic expert system. *IEEE Computer Special Issue*, 17(9):51-58, September 1984.
- [26] R. S. Patil, P. Szolovits, and W. B. Schwartz. Causal understanding of patient illness in medical diagnosis. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence*, pages 339-360. Addison-Wesley, Reading, MA, 1984.

- [27] H. E. Pople. The formation of composite hypotheses in diagnostic problem solving. In *Proceedings of the 5th IJCAI*, pages 1030-1037, Cambridge, MA, August 22-25, 1977.
- [28] W. F. Punch III, M. C. Tanner, and J. R. Josephson. Design considerations for PEIRCE, a high-level language for hypothesis assembly. In *Proceedings of the Expert Systems in Government Symposium*, pages 279-281, McLean, VA, October 22-24 1986.
- [29] J. Reggia. Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, 19(5):437-460, November 1983.
- [30] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57-95, April 1987.
- [31] A. C. Scott, W. J. Clancey, R. Davis, and E. H. Shortliffe. Methods for generating explanations. In B. G. Buchanan and E. H. Shortliffe, editors, *Rule-Based Expert Systems*, chapter 18, pages 338-362. Addison-Wesley, Reading, MA, 1984.
- [32] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In J. L. Kolodner and C. K. Riesbeck, editors, *Experience, Memory and Reasoning*, pages 47-73. Erlbaum, Hillsdale, NJ, 1986.
- [33] J. W. Smith, Jr., J. R. Svrbely, C. A. Evans, P. Strohm, J. R. Josephson, and M. Tanner. Red: A red-cell antibody identification expert module. *Journal of Medical Systems*, 9(3):121-138, 1985.
- [34] J. Sticklen. *MDX2. An Integrated Medical Diagnostic System*. PhD thesis, The Ohio State University, Columbus, OH, 1987.
- [35] W. R. Swartout. XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3):285-325, September 1983.
- [36] M. C. Tanner. *Explaining Knowledge Systems: Justifying Diagnostic Conclusions*. PhD thesis, Dept. of Computer and Information Science, Ohio State University, Columbus, OH, March 1989.
- [37] J. L. Weiner. BLAH, a system which explains its reasoning. *Artificial Intelligence*, 15(1/2):19-48, November 1980.
- [38] M. R. Wick, W. B. Thompson, and J. R. Slagle. Knowledge-based explanation. TR 88-24, Computer Science Dept., Univ. of Minn., Minneapolis, MN, March 1988.
- [39] T. Winograd. *Understanding Natural Language*. Academic Press, New York, NY, 1972.

Appendix B

Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions

Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions*

Anne M. Keuneke

July 30, 1989

Abstract

This research investigates how diagnostic conclusions made by a diagnostic problem-solving system can be explained by showing how the hypothesized malfunction causally gives rise to the observations. In particular, it shows how this explanation can be constructed from a "deep" model of the system being diagnosed, i.e., a model that explicitly represents the underlying structure of the system and the functions and behaviors of its components.

In the past, researchers have investigated how diagnosis itself can be performed from deep models, but using these representations for individual diagnostic problems can be computationally expensive. Compiled knowledge helps in efficiency in problem solving. However, the compiled knowledge may be incomplete and there may be a doubt about the correctness of the diagnostic conclusion. If a causal story can be put together computationally effectively, using the diagnosed answer as a focus, one gets the computational benefits of compiled knowledge to obtain the diagnostic answer, as well as the use of the deep model for causal validation and elaboration of the answer.

The goal of this research is to determine how, and to what degree, causal explanations of malfunctions can be derived from an understanding of how a device is expected to work. Efforts simultaneously concentrate on the following: (1) what it means to provide a causal story of device function and malfunction, and (2) in order to produce these stories, what knowledge is needed to model devices and how should it be organized and indexed in a way that contributes toward a general solution to the representation of device.

The representation developed in this work builds upon an earlier model introduced by Sembugamoorthy and Chandrasekaran called the Functional Representation. Representational enhancements include the specification of a taxonomy of function types (achieving a state, maintaining a state, preventing an undesirable state, and controlling variations in state), and explicit representation of state and behavioral abstractions (such as oscillation and feedback). The explanation generation process then uses the organization and primitives of the representation to simulate malfunctioning devices and produce causal chains, consisting of malfunctioning components and observations, leading from diagnostic hypotheses to observed symptoms.

*This work has been supported by the Defense Advanced Research Projects Agency under RADC contract F30602-85-C-0010.

Chapter 1

Introduction

The primary goal of diagnosis is to determine what malfunction is causing the observed symptoms. This task can be done with compiled knowledge [1], i.e., knowledge that directly relates observations to malfunction hypotheses. Once a diagnostic answer is obtained, it may remain unclear as to how the hypothesis, or hypotheses, actually explain the symptoms. An expanded causal story, from malfunction hypothesis to observations, could provide this connection and add to one's comprehension, but this knowledge is often not accessible in compiled form. In fact, such storage would be combinatorially explosive: there are simply too many possible causal paths. Human reasoners are capable of generating these causal stories using a basic understanding of how the malfunctioning device works, i.e., knowledge of how the structure of the device relates to its functions and behaviors. Representations providing this knowledge have been called deep models [1].

Deep models have been used to perform diagnosis itself [2, 11, 13], but for complex devices working directly with deep models involves dealing with large amounts of causal information, unnecessarily increasing the complexity of the diagnostic task. Typically, the knowledge available through causal models is condensed and used in producing compiled systems for greater expertise and optimum performance for diagnosis. Then, if the causal story can be put together computationally effectively, using the hypothesized diagnostic answer as a focus, we get the advantages of both worlds: the computational benefits of compiled knowledge to obtain the diagnostic answer, as well as the use of the deep model for a causal validation of the answer.

This research proposes a method for generating causal justifications of diagnostic conclusions, based on a functional representation of devices, which illustrates how an expert's understanding of the functioning of a complex device is related to its structure and behavior. The object of the work is to determine how, and to what degree, causal explanations of malfunctions can be derived from an understanding of how a device is expected to work. Efforts have simultaneously concentrated on the following:

- What is required to provide a causal story of device function and malfunction?
- In order to produce these stories, what knowledge is needed to model devices and how should it be organized so that it contributes toward a general solution to the representation of devices?

In the global sense, i.e., the theoretical contribution to Artificial Intelligence, this research is intended to make progress towards full machine understanding of devices. The usefulness of causal models currently

available in AI systems is often stymied due to the lack of extensibility. Such representations often suffer from the "scaling up" problem when faced with different types of devices or when representing complex systems. In this work, I demonstrate the potential of a functional model, based on earlier work of Sembugamoorthy and Chandrasekaran [14], by showing how such an organization, together with the additional enhancements proposed in this research, provide a model which allows for expansion to serve the needs of various problem-solving tasks and also for the representation of certain complexities of device understanding. First, I give a brief description of the representation proposed in [14].

1.1 The Functional Representation

1.1.1 Overview

The functional representation combines the following information about a device:

Structure: specifies the components of a device and the relations between them.

Function: specifies *what* is the result or goal of an activity of a device or component.

Behavior: specifies *how*, given a stimulus, the function is accomplished.

General Knowledge: pointers to general knowledge of the domain that show how key states occur. This is the "bottoming out" knowledge for when it is not desired to expand a casual transition to further behavioral or component detail. More specifically, this is the body of knowledge (such as the laws of physics and the underlying equations or empirical or statistical knowledge) that may be invoked to explain or justify a causal transition.

Assumptions: assumptions under which a behavior is accomplished. These are expectations about certain behavioral states that are commonly made when certain causal transitions occur, e.g., liquid will flow through a pipe *provided* either the source has a higher elevation than the output, or the flowrate is great enough to force it through.

This information is combined with the perspective that an agent's understanding of how a device works is represented in a way that shows how an intended function is accomplished as a series of behavioral states of the device. The device itself is represented in various levels. The topmost level describes the functioning of the device by identifying which components and more detailed behaviors are responsible for bringing about the various state transitions. If a transition is achieved using a function of a component, the next level describes the functioning of this component in terms of the roles of its subcomponents, and so on. Ultimately, either by tracing through more detailed behaviors or by expanding the behaviors of functional components, all the functions of a device can be related to its structure and the functionality of the components within this structure.

The Buzzer Example

In the following I use the household buzzer, the example used in [14], to explicate the primitives of the functional representation. Although with later enhancements to their representation I change the model of this particular device somewhat, in this section I describe their work and thus illustrate with their primitives

to represent the functioning of the buzzer. What is important here is the organization and basic language of primitives — these are not changed, simply expanded.

1.1.2 Structure

Figure 1.1 provides a physical and functional sense of the structure of the buzzer. For visual clarity and later reference, the physical structure is shown in the form of a schematic diagram. The device-function browser illustrates the functional components and the functions associated with them. When devices are simple there is often no difference in functional or physical structural component specifications. For a discussion of the distinctions between functional and physical structure specifications see [7].

1.1.3 Function

Devices are functional objects. In a functional representation, the structure of the device is built using functional components. Thus every component is associated with a function or functions, as seen in the figure. In understanding a function, the information required includes: what is the goal, how can this goal be achieved, and when would the achievement of this action be desired, i.e., what is its starting state. These "definitional" primitives, i.e., those which identify a certain functionality, are characterized through the primitive terms: ToMake, By and If.

The functional specification of a buzzer is illustrated by describing its function, to *buzz*, using these primitives.

FUNCTION: Buzz

ToMake: (Buzzing Buzzer)

If: (Pressed ManualSwitch)*

By: behavior1

Provided: assumption1

In this description, (Buzzing Buzzer) and (Pressed ManualSwitch) are partial state descriptions. The * is meant to indicate repetition of state. The By clause is a pointer to a behavior that accomplishes this functionality. Notice that the entire behavioral specification is not stored here. This same function could be achieved in a number of ways, the identification of behavior1 indicates how this functionality is achieved within this specific component. This separation is important when an agent needs to replace a malfunctioning component by a functionally equivalent but a behaviorally different one. The Provided clause specifies the assumptions under which this specific behavior is accomplished. It is, of course, impossible to give all state descriptions that will hold when any behavior is being realized, but, *ceteris paribus*, there are certain preconditions and expected states that are commonly known to be of interest. Assumption1 indicates that initially t7 and t8 are electrically connected (see structure diagram).

1.1.4 Behavior

Behavior is represented by a sequence of partial states with the specification of *how* each state transition occurred. The behavioral specification of a device describes the manner in which a function is accomplished using three conceptually important notations. A state transition can be accomplished by the following:

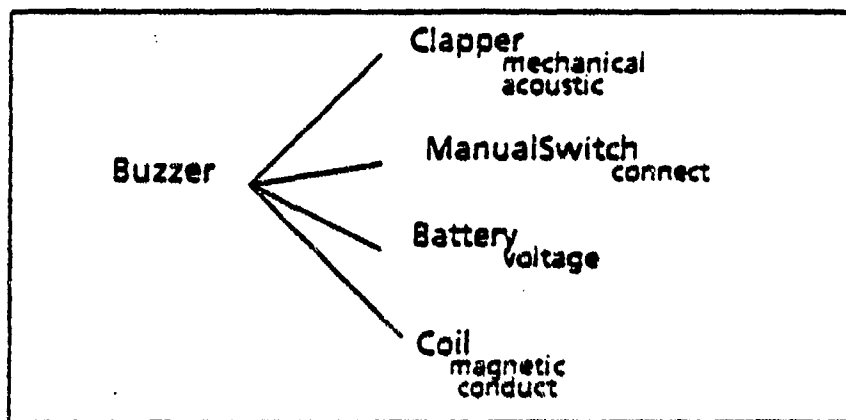
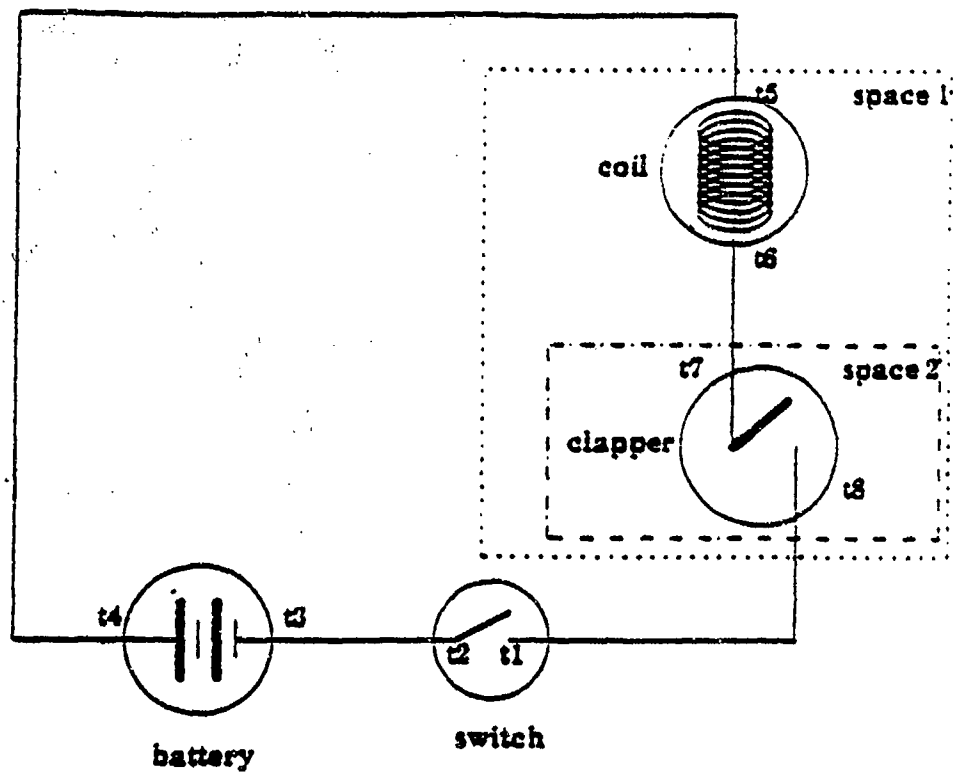


Figure 1.1: Structural Representations of a Buzzer

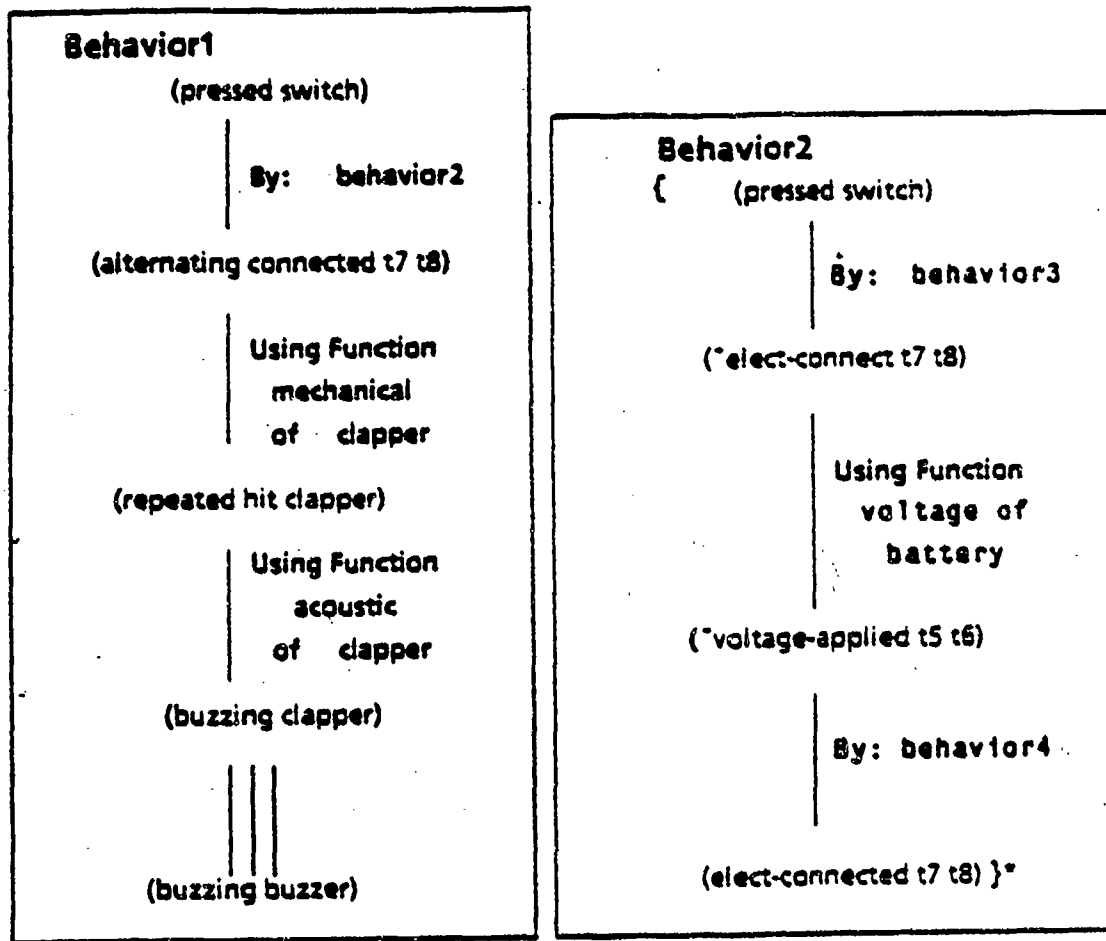


Figure 1.2: Behavior1 and Behavior2 for function Buzz

UsingFunction:: specifies that this state transition is accomplished through the use of a function of some component of this device

AsPer: specifies general knowledge which can explain the transition

By: indicates that this transition is explained in greater detail by the specified sub-behavior.

The behavior for behavior1 of the *buzz* function is described by a chain of events brought about by the specified actions as seen in figure 1.2.

The figure is meant to represent the temporal sequence (from top to bottom) of states that occur as a result of actions taken. "Behavior1" indicates that the Buzzer, on the occasion of the switch being continuously "Pressed", goes to a state where the electrical connections in the clapper alternately close and open, which results in the state where the clapper is repeatedly hit, which results in the Buzzer being in the state of

"Buzzing". Each transition is further explained, either in terms of further details in the state transition, or in terms of the functions of the components. For example, the transition from the clapper being alternately electrically connected and disconnected, to its being in the "repeated-hit" state, is explained by relating it to the "mechanical" function of the clapper and inspecting the behavior for this function.

The specification of how the state transitions occur through the use of the UsingFunction, By and As Per constructs provides the representation with the capability to express both a hierarchy of detail and a component hierarchy. The By construct allows the representation to provide more detail regarding how the state transition occurred. By tracing behaviors for the Using Function construct, it is apparent which subcomponents are used to perform the function. Both hierarchies can thus be expanded by delving deeper into a function's behavior. When one has reached the bottom level of detail and component specification desired, transitions are explained using As Per.

A Partial Coverage

The above description is far from a complete coverage of the buzzer's representation. For more detail see [14]. This partial description has covered the basic conceptual points of interest, i.e., the organizational scheme and primitives specified by the representation of Sembugamoorthy and Chandrasekaran.

1.2 Research Methodology

The approach taken in this research centers on the hypothesis that causal explanation of malfunction can be derived from device representations organized around expected functionalities. A solution to this explanation derivation involves contributions toward both a general solution for device representation and a methodology for the generation of causal malfunction sequences.

1.2.1 Representation of How a Device Works

First, a model must be developed which incorporates the understanding needed for explaining how a device works. This involves the determination and representation of needed primitives, and the organization of this knowledge for accessibility. In this research, the functional representation just described was used as a foundation. This representation alone, however, did not sufficiently model various aspects of complex devices. Insight about additional desirable knowledge was obtained from two sources. First, by modeling a number of simple and complex devices, including a chemical processing plant, the need for explicit representation of constructs such as expected side effects, cyclic behaviors and feedback mechanisms was empirically observed. Second, by focusing on the task of explanation, the need for more contextual distinctions, specifying *why* actions were taken or functions used, was determined. In addition, the necessity to model behavior from different perspectives was noted.

In the following, I show how the functional representation is enhanced to accommodate these needs by the addition of explicit knowledge constructs to the functional and behavioral specifications of device.

1.2.2 Explanation of How a Device Fails to Work

Provided with the knowledge of expected functions and the behaviors which realize these functions, together with a knowledge structure which makes explicit the interactions between various components, the explanation system can derive a causal sequence of malfunctioning components. In the final sections, I distinguish between types of consequences of malfunctioning units, and discuss the capabilities of the adapted functional representation to provide the necessary knowledge to trace these consequences. For the malfunction explanation which can be derived directly from the functional representation, I describe the reasoning used to generate the causal chain. In addition, I discuss what additional knowledge can be accessed for an even richer explanation.

Chapter 2

Enhancing the Functional Representation

The technique used in this thesis to generate causal explanation of diagnostic conclusions centers on the hypothesis that causal explanation of malfunctions can be derived from a representation organized around the expected functionalities of a device. The functional representation described earlier provided an organizational kernel for representing knowledge of expected functionality, but to represent and explain the mechanisms of complex devices, this core alone is not sufficient. This chapter enhances the functional representation by supplying additional distinctions about functions, behaviors, and states.

2.1 Taxonomy of Function Types: Distinctions of Purpose

In the following I identify four distinct function types. Each function type indicates different procedures for simulation, different functional capabilities, different expectations, and thus different knowledge specifications for representation and explanation. I discuss the knowledge which distinguishes each type, provide explicit specifications, and describe the information processing distinctions each new type provides. Goal types include:

1. **ToMake:** achieves a specific partial state
2. **ToMaintain:** achieves and sustains a desired state
3. **ToPrevent:** keeps a system out of an undesirable state
4. **ToControl:** gives a system power to regulate changes of state via a known relationship.

2.1.1 ToMake

Semantics

The function type of ToMake is the basic functional type. The idea here is that a component's function is either to achieve a specific state of the device, as in the use of a bolt on a door ToMake: (door locked), or to achieve a value or state of some substance parameter that the device manipulates, i.e., the function of a condenser is ToMake: (substance liquid). In general, the objective of a ToMake function is the achievement of a specific partial state.

Specification

The primitives of this functional type are the basic definitional primitives:

FUNCTION:

ToMake: (desired state)

If: (action/state for initiation of the function's behavior)

By: (identification of behavior which achieves the function)

Provided: (assumptions under which the behavioral states are valid)

2.1.2 ToMaintain

In contrast, consider the functionality of devices meant ToMaintain certain states. ToMake functions achieve states which specify a parameter value and are true or false at a given time, e.g., to achieve (liquid temperature 40°C). ToMaintain functions achieve states which indicate that a process is actively sustaining the parameter at that value over a period of time, e.g., (maintained liquid temperature 40°C).

Semantics

In general, maintenance implies three important features: there exists continuous monitoring, there exists a range within which a variable is expected to be maintained, and there exists potential for the adjustment needed to keep the variable value within that range — all three combine to form one functional unit in order to achieve and sustain the desired state. Behaviors for maintenance are not simple linear causal sequences which reach an end state; rather, they involve continuous cyclic action to *keep* the system in some desired state.

Specification

The representation for maintenance functions which specifies the knowledge just discussed is as follows:

FUNCTION:

ToMaintain: (substance/component variable)

Range: (desired interval or value)

If: (initial state)

By.ContinuousMonitor: (behavior to monitor and trigger)

Provided: { assumptions }
 Until: { triggering situation }
 By.Adjustment: { behavior to adjust }
 Provided: { assumptions for By.Adjustment }
 NotBehavior(s): { behavior for what would happen if not maintained }

Using the specifications given, a house heating system can be represented as follows:

FUNCTION:

ToMaintain: (air temperature)
 Range: (70-73°)
 If: (Thermostat On)
 By.ContinuousMonitor: ThermostatBehavior
 Provided: (Thermostat.Sensor activates within.RangeSetting)
 Until: (temperature air LessThan.Range)
 By.Adjustment: FurnaceBehavior
 Provided: (Range NOT(GreaterThan) FurnaceCapacity)
 NotBehavior(s): ColdHouseBehavior

This information indicates that the function of the heating system is to maintain the temperature of the air to between 70 - 73°. *ThermostatBehavior* points to a behavioral sequence which shows how the thermostat is used to monitor the temperature until it is less than the range value, at which time it triggers the furnace. Once the furnace is activated, it heats the surroundings to reach the desired temperature. This behavior is accessed through *FurnaceBehavior*. Because this function is a maintenance function and the monitor mechanism is continuous this behavioral sequence will be repeated.

NotBehaviors

Access to knowledge of *how* the function is achieved is provided through the functional *By* specifications. For the purposes of explanation, it is also useful to specify *why* the goal is desired in the overall system. For maintenance functions, knowledge of these aspects (how and why), are represented in varying contextual levels. Specifically, although the detailed behavior for *how* the component achieves its function can be viewed at a level relative to the component, the behavior to illustrate *why* it is used is not helpful unless viewed from a higher perspective of the overall system.¹ For maintenance functions, knowledge of why the function is used typically indicates what would result to the system if this function were *not* used. This knowledge is specified using the *NotBehavior* primitive.

The *NotBehavior* may have multiple behaviors identified or it may have none, depending on the problem-solving task for which the device representation is used. Since substances are maintained within a range, one may want to specify what significant events occur at either end of the interval if the system malfunctions. For example, what happens when the temperature continues to be hotter than 73° or when the temperature continues to be lower than 68°.

¹A substance is maintained at a value for a reason in some higher context. Representing this knowledge follows the concept discussed by Kuipers [9] (and in [7]), that information should be modular to the unit itself, but if a larger story is known and needed then the information should be used.

Information Processing: Distinctions and Uses

The primary distinction between ToMake and ToMaintain functions is that of intended temporal persistence of state. Devices designed to achieve maintenance require components and mechanisms whose effects produce both the achievement and persistence of the state. Problem-solving tasks, such as simulation, diagnosis, prediction, design, and repair, must address issues involved in providing this continuity of behavior, and also the consequences of continuous behaviors, when considering devices with maintenance functions.

For the task of device explanation, the explicit representation of aspects of sustaining a state, i.e., monitoring, range values, and adjustment, allows the explainer to define and distinguish the integral role that each aspect plays toward keeping the device in the desired state. For example, individual components may be working but the device is not providing the appropriate effects because the interrelationships between the components were not properly set. By identifying a function as maintenance and providing specifications for the characteristics which define it, one has the potential to focus on the higher level issues involved in maintaining the state.

For instance, suppose a home heating system is not maintaining the desired temperature of 70°. Responses to the question of, "Why is the temperature 75°?" could include any, or all, of the following:

- The thermostat was set at 75° (established Range). The system is behaving as expected given this setting.
- The monitor's mechanism (in By.ContinuousMonitor), which is meant to monitor the temperature and trigger the furnace, turns on the furnace at a temperature value (defined Until) other than that set by the thermostat (defined Range).
- The switch on the furnace (a component used in By.Adjustment) gets stuck and does not respond immediately to the signal from the triggering device. Thus the temperature adjustment is higher than the range desired.
- The heating system maintains a temperature of 70° by heating the room to a temperature of 75° and allowing the hotter air to mix with the colder air of the room, thus adjusting to a temperature of 70° (mechanism in By.Adjustment).

Additional malfunction explanation is also available through the NotBehavior clause. This clause gives the explainer the potential to explain symptoms due to the malfunction in a higher behavioral context, e.g., effects in the environment.

2.1.3 ToPrevent

The third function type is ToPrevent. ToPrevent is similar to ToMaintain in that global goals appear the same. For prevent functions, the function's behavior explicitly prevents an undesirable state, as opposed to maintenance behaviors which continuously provide a desired state. The primary differences are that of intent and continuity of use in a given system. ToPrevent functions provide short-term fail-safe mechanisms, not operations performed for the normal and continuous maintenance of a system.

Functional Distinction: ToPrevent vs ToMaintain-NOT

For an example to better distinguish the types, consider the following: The dikes in Holland maintain the state of keeping Holland dry. Once a hole developed in this wall, there was a threat. The famed boy prevented Holland from flooding and disaster by putting his finger in the hole. The boy achieved the function ToPrevent: (flood). The wall was failing to achieve a function ToMaintain: (NOT flood).

One might argue that for the walls, a functional specification of ToMaintain: (NOT flood) is equivalent to ToPrevent: (flood). Occasionally these functional specifications may be logically equivalent, but not always. In the above situation, one would not expect to see individuals along the wall daily ToMaintain a safe environment. Specifically, the boy prevented a flood, he had no intention to remain at the wall in order to achieve the continuous action necessary for ToMaintain: (NOT flood). Thus ToPrevent \neq ToMaintain-NOT. On the other hand, consider the lights in a building. Their function is ToMaintain: (illumination), or ToMaintain: (NOT dark). Their function is not ToPrevent: (dark). Thus, ToMaintain-NOT \neq ToPrevent. The constructs are not equivalent or reducible.

Specification

To illustrate the specification of the ToPrevent function type, I represent the function of a rupture disk in a chemical processing plant. A rupture disk is a metal disk, with an appropriate tensile strength, attached to the reaction vessel. Its function is ToPrevent the reaction vessel from blowing up. That is, if the pressure is too high in the vessel the disk will blow, thus releasing steam and reducing pressure. When the disk blows, the release of pressure prevents the entire vessel from destruction.

FUNCTION:

ToPrevent: (reactionvessel blowup)

Threshold: (tensilestrength value)

If: (rupturedisk present)

When: (pressure reactionvessel at.disk.threshold)

By: behavior blowout

Provided: assumptions for blowout

NotBehavior(s): behavior NotBlowup

The primitive When indicates the situation which triggers the use of the device; When the prevent function is triggered, the behaviors blowout and NotBlowup are observed. Behavior blowout shows the chain of events in terms of the disk itself. For instance, it shows how structural stress degradation in the disk leads to a state of (disk ruptures). Note that such information does not provide knowledge of what was prevented, i.e., it does not show the behavior which illustrates the *purpose* of the device in the overall system. The NotBehavior NotBlowup is used to specify the view from the system: (disk ruptures) \rightarrow (escape steam) \rightarrow (reduce steam) \rightarrow (reduce pressure) \rightarrow (NOT (reactionvessel blowup)).

Information Processing Distinctions

Prevention devices are not used during normal functioning of a device (e.g., fire extinguishers). As opposed to maintenance functions, their use is not observed in the normal causal flow of events. Simulation of a

device involves ToPrevent functions only when the contingency arises. The use of these functions requires the processing capabilities to "watch" for the triggering situation.³

In addition, devices with ToPrevent functions have different design considerations. Devices used for prevention are often not designed for continuous or repeated use. In fact, behaviors used typically require the use of something (or someone) which is expendable (or at least not repeatedly available). Identification of such commodities is useful for repair, replacement, or reuse of the function, in addition to simulation. For example, once the rupture disk blows, the chemical plant will not continue normal processing until the disk is replaced.

2.1.4 ToControl

The last functional type is ToControl. In the ToMake, ToMaintain, and ToPrevent function types, the expected goal was the achievement of some state (either a predicate or a process state). Control indicates power to regulate. To control something, or someone, means that there exists a direct multivalued relationship between the device's action and the resultant effects.

Specification

To illustrate, consider the (inexact) specification of the function of a faucet.

FUNCTION:

ToControl: (water flowrate)

Relationship: flow is proportional to handwheel position via function $f(x)$

Options: rotation x (off) \rightarrow rotation x (maximum on)

If: (present water valve)

By: behavior `adjust.water`

Provided: assumptions for `adjust.water`

NotBehavior: `Handwheel.Stuck`

The behavior `adjust.water` is illustrated in figure 2.1. The functional primitives provide knowledge that the faucet is used ToControl the flowrate of water. The Relationship primitive states the expected control relationship. For the faucet, it gives the proportionality between the rotation of the handwheel (turned handwheel rotation x), and the expected amount of flow (flowrate water valve y). The Options primitive indicates the available range of values for the control mechanism. Depending on how far the handwheel is turned, the function specifies the expected amount of water according to the Relationship. If there is no water at the valve, or if the handwheel is not adjustable, or if the handwheel is not turning the stem to move the valve as it should, then the valve can not control the flow. NotBehaviors specify what happens if the valve is stuck, or other conditions which indicate the loss of control capabilities.

Information Processing Distinctions

The knowledge specific to control functions is the capability to provide multiple outputs and the ability to manipulate components in order to provide the exact output desired. Simulation of control functions requires

³This is accomplished through what is commonly called *active values*.

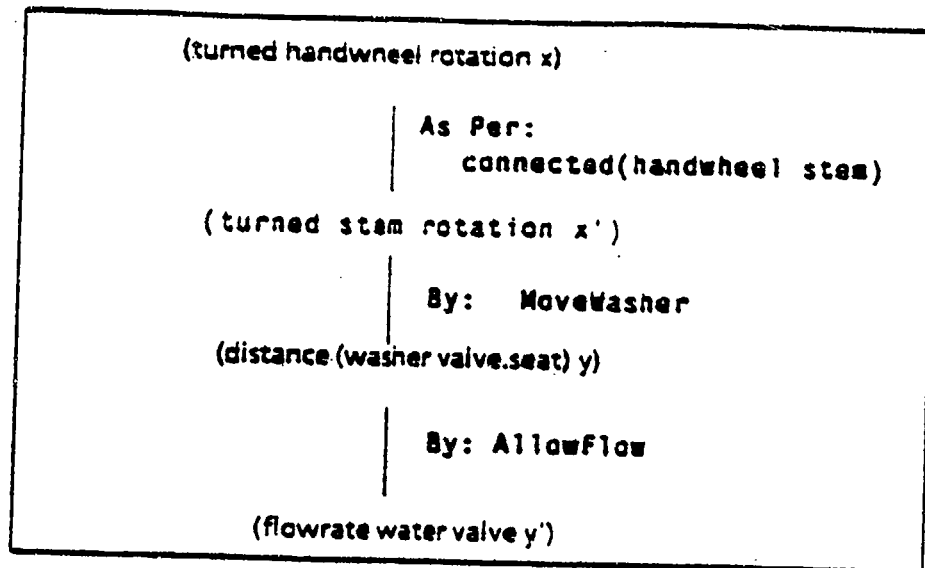


Figure 2.1: Behavior adjust.water

the capability to adjust the behavior (specify state parameters) in a way to illustrate how the given state causes the end result. Specifically, the parameters inside the behavior change due to the input setting (within options) of the device and the given relationship. The design of devices which expect to control must also take this relationship into consideration.

Explanation of malfunctions in control functions includes not only checking the general behavior that leads from initial state to final state, but also considering malfunctions of aspects of control. For a device to have the function of controlling, there are two necessary requirements. The first is that there exists the capability to make adjustments. For instance, a faucet no longer controls water flow if the valve becomes stuck, the handwheel is no longer adjustable, or if there is no water to regulate. These situations are represented in NotBehaviors. The second requirement for control is that the proposed adjustments must cause the specific expected results, i.e., the relationship must remain valid. If there is a hole in a valve of the faucet, or the handwheel is not turning the stem to move the valve as it should, the expected flow will not be achieved; the designed relationship between the handwheel and flowrate is not valid.

Functional Distinctions

A water faucet is an excellent example for illustrating the differences between ToControl and ToMake or ToMaintain functions. Consider the faucets which bounce back if one does not hold onto the handle. Here the device achieves a state (as in ToMake), but it also has the ability to produce a multitude of states if desired (as in ToControl). The control functional specification provides the knowledge of these potential states through the Relationship primitive. This type of faucet shows how control does not necessarily indicate maintenance. Specifically, even though the faucet provides control, extra mechanisms would be needed if it is desired to keep the handle, and thus flowrate, at a specific value. Alternatively, having the ability to maintain does not imply that the device has the capability to regulate or control. There also exist

faucets which only maintain a single flowrate with no control capabilities.

2.1.5 Problems: Proliferation and Completeness

In the previous sections, I proposed a taxonomy of function types. As with any delineation to establish a set of primitives, this collection is subject to problems concerning proliferation and completeness. Although in these sections I showed distinctions between the functional types to illustrate variations in the constructs, I make no claims as to the completeness of this set and concede that any specification of a set of "primitives" for knowledge representation can be questioned as to whether it can be reduced. However, I believe that this set represents core aspects of functional understanding and justify their specification on the grounds of utility for explicit representation for inference making during problem solving.

2.2 Additional Functional Distinctions

In the next two sections, I present functional taxonomies orthogonal to that of function type. Each section presents a new distinction and primitive to provide insight on when a device is, or is not, achieving its functions.

2.2.1 Passive and Active Functions

Active functions indicate the capability or involvement of the component *itself* to cause movement, action, or force. In such situations, the function's detailed behavior describes how the component or parts contribute to the action.

Sometimes devices achieve their functionality by simply being present because the structure of the device has a necessary property which allows it to perform some duty. I call this type of function a **passive function**. For example, the functions of the reaction vessel in a chemical processing plant are to maintain a "closed" system and to provide a contained environment in which the reaction can occur. The vessel does not actually do anything. It achieves these functions by simply being present and not allowing leaks. This is accomplished through its structural integrity, i.e., by having the attribute that it can withstand a certain amount of pressure. Other examples of passive functions are walls, windows, door stops, paper weights, chairs, etc.

Specification

The representational differences between active and passive functions involve (1) variation in the context level of the behavior specification and (2) an added primitive to indicate important traits.

A behavioral specification of "how the device works" for passive components can only be explained by considering the expected use of the device in the greater context. Passive functions often need outside initiators or actions to motivate their use. A hammer can be used to do things because of certain aspects of its structure, the hardness of its head and the force provided through use of the handle. Here the hammer does not actively participate, it is not producing action, instead some outside action is needed to take advantage of its function. Behaviors for functions which are passive do not indicate internal behavior of the device, but instead refer to actions of the initiator or user.

Specification of the attributes of importance, i.e., those which allow the component to provide the necessary results, allows for the identification of other "hammer-like" devices which can perform the function. Passive functions require an extra primitive **PassiveProperties** to provide this specification of important characteristics of structure.

The specification of the function conduit for a pipe, used to transport liquid, could be represented as a **ToMake** function as follows:

FUNCTION: conduit

ToMake: (location liquid x)

If: (location liquid y)

PassiveProperties: tubular, nondissolvable by liquid

By: flowbehavior

Provided: (OR (sufficient force flow)(lower location x location y))

Depending on the topology of the overall device and the availability of knowledge about the substance which is being transported, the flowbehavior can vary in complexity. It could be trivial, i.e., simply (location liquid y) \rightarrow (location liquid x), or it could use knowledge of the pipe angle and liquid viscosity to be precise regarding flow rates.

Information Processing Distinctions

Passive functions are not a new functional type; there is not a new kind of goal, but instead a different explanation of behavior through which the device achieves its goal. It achieves a desired function by being at the right place at the right time with the right characteristics. Simulation of passive functions requires the capabilities, not to simulate the functional component's behavior, but rather to simulate the substance or component changed as a result of using the function.

When considering replacement of passive functional components, one looks for components with the same desirable traits. The specification of these traits is also useful for explaining why a device is not achieving its desired function. If the function is accomplished due to a certain trait, and the device is flawed with respect to this trait, the function may not be achievable.

2.2.2 Primary and Secondary Functions

Often devices are intended and designed for a particular primary goal but include additional functionalities for various reasons (such as safety, economy, aesthetics). In the analysis of device functionality, one observes that some functions have a secondary nature. These are functions which are present in support of another main function. Three types of secondaryness are delineated:

- Subfunctions:

- functions a device possesses simply as a means to establish preconditions for a primary function. For example, the only reason an aircraft has the function "takeoff" is so that it can achieve the function "to fly".³

³ It might be argued that any function in a behavior could be considered to be setting up preconditions for the functions that follow or that subfunctions could be represented by having all functions have precompiled "used-by" primitives globally. The idea here is

- functions a device possesses to support a provided clause (assumptions) on behaviors for a primary function. A car can be used for transportation provided the driver can see the road. The functionality of windshield-wipers allows for this behavior in inclement weather; the functionality of lights provides sight at night.

- **Auxiliary functions:**

With respect to the desired use of the given device, these are extraneous functions. Consider antiques. An antique chair could be used for sitting, or simply for the beauty of the room, or for both. Auxiliary functions could be primary functions depending on the device designer's or user's purposes.

- **Other design considerations:**

Design considerations are goals which arise out of the situation or context in which the device is used. Generally these focus around additional constraints on how the designer wants the main function to be achieved, i.e., economy, aesthetics, ecology. These could also arise as safe-guards to compensate for undesirable side effects of behaviors. For example a lawnmower's protective shield or the wheels on the legs of a chair have functionalities. With respect to the main goal of the lawnmower or chair these components are irrelevant. Explanation of why the functionalities are needed involves "external" considerations. The devices have these functionalities because when using the device, the user also has the desire to protect himself or move about easily.

Specification

An example of a secondary function in a chemical processing plant is the *send.signal* function of the sensor/alarm system. Its functional description is as follows:

FUNCTION: *send.signal*

ToMake: (present alarm signal ProgrammableController)

If: (temperature sensor above.threshold)

By: signal behavior

Provided: ()

SubFunctionOf: PressureMaintenance

External Consideration: safety (to identify location where overheated)

Information Processing

The *send.signal* function is identified as a SubFunctionOf PressureMaintenance. Use of *send.signal* is not seen in a trace of the main causal chain or detailed levels of the behavior for the plant. However the PressureMaintenance function has a Provided clause indicating that the function *send.signal* must be working. The idea is that the pressure can be maintained as long as problem areas can be identified in time in order to compensate. The *send.signal* function must achieve this goal. The SubFunctionOf specification identifies the function whose Provided clause must be accomplished, and thus assists in the proper simulation

that given devices have functional components which are "packaged". We want this grouping of component functionality as an aid for exportation of function and the knowledge to interchange components. By using subfunctions for components within a device one can identify the main purpose of the device and also capture the degree of component modularity desired.

of the plant. In addition, it is used for determining a more complete malfunction chain, i.e., a malfunction here results in problems for the PressureMaintenance function.

The send.signal function is also identified as an external consideration. Here it is identified as a safety precaution. With the External Consideration stipulation, explanation of malfunctions has available the knowledge that the overall system would still work without this device present, but not as safely.

In general, specification of secondary functions is necessary to represent understanding of why the function is present in the device and for determining equivalency when considering replacement of components.⁴

2.2.3 Summary of Functional Enhancements

The functional specification has now been enhanced to provide the following: four distinct functional types, ToMake, ToMaintain, ToPrevent, and ToControl with their related primitives. In addition, the primitive, PassiveProperties, is present to indicate passive functionalities, and the SubFunctionOf and ExternalConsideration primitives provide knowledge of secondary functions. With these additions the explanation capabilities to describe what is the function, how it is achieved, and why the function is desired become more complete.

2.3 Representing Behavior: Knowledge of How and Why

The behavioral representation in the functional representation consisted of a causal sequence of partial states, linked together with the identification of *how* state transitions in behaviors were expected to occur using the notions of UsingFunction, By behavior, and AsPer knowledge. The understanding and explanation of behavior could be enhanced through following information: why was the action performed, to what extent does the action have to be completed for accomplishment of the goal within this device, and what side effect behaviors are important to the proper functioning of the device.

The following subsections describe the additional primitives to represent this information. However, since the original behavioral specification provided in the functional representation addressed only linear behaviors, I would first like to illustrate the representation of behaviors which require multiple causal chains. Section 2.4.1 discusses the representation of cyclic behaviors.

2.3.1 Behavioral Representation: Linear Causal Links

Representing Parallelism in Behavior

Often a desired state can be a conjunction of partial states. These states may not be causally related, thus the behavior to achieve the conjunction requires multiple causal paths, one for each conjunct. For example, the state of (prepared aircraft) may be equivalent to (AND (loaded passengers) (loaded baggage)(loaded fuel)). A behavior *Prepare.Craft* which represents how (prepared aircraft) is achieved is illustrated in figure 2.2. Accomplishment of all of these states indicates that the aircraft is prepared. The simulator interprets multiple links as an AND clause and performs each. Because the partial states are not causally related, the order of performance is insignificant; the actions can be performed in parallel.

⁴Because knowledge of auxiliary functions does not provide information on how the device in question works, a primitive for specifying this knowledge is not provided here. Such a specification is reasonable for tasks such as planning or design where the multiple uses of the device could aid in the determination of which device to choose.

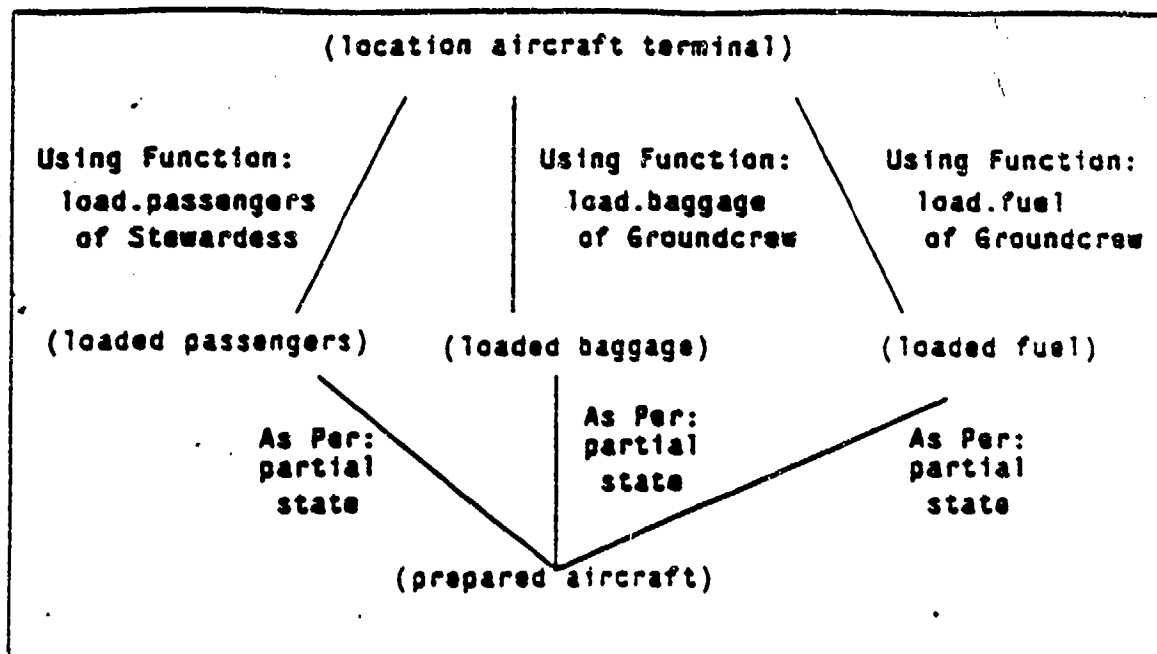


Figure 2.2: Behavior Prepare.Craft

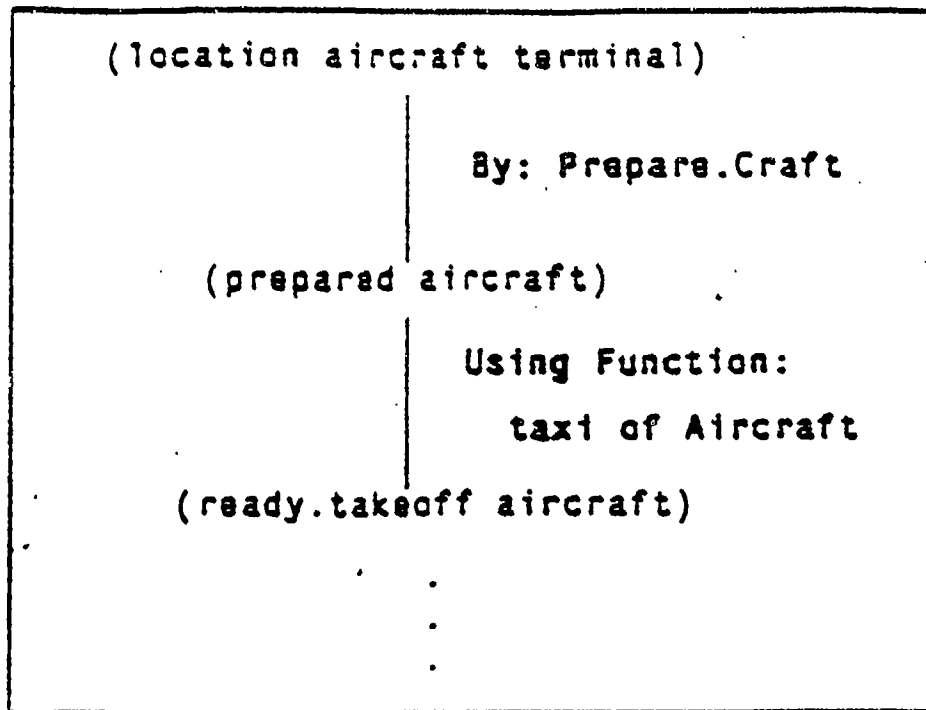


Figure 2.3: Behavior PreFlight

Equivalence of State

The knowledge that the state (prepared aircraft) is equivalent to (AND (loaded passengers)(loaded baggage)(loaded fuel)) is identified by the specification of *AsPer: knowledge of partial state*. Equivalent states which are not conjunctions can be represented in a similar fashion. i.e., *AsPer: identity*.

2.3.2 Identifying Behavior Fragments With Roles

The behavioral specification to illustrate how a function was achieved was represented in the functional representation through causal chains where the links were annotated with knowledge of *how* the state transition occurred. Knowledge of *why* that state transition was used within the behavior was implicit. The assumption was that a behavior is used to achieve the next state in the causal chain. However, actions are often performed in behaviors for reasons not explicit in the immediate behavioral specification.

For instance, consider preparing an aircraft for takeoff. Part of the behavior showing the departure of an aircraft is shown in figure 2.3. The *Prepare.Craft* behavior was shown in figure 2.2. Descriptions of behaviors showing how these actions were achieved do not indicate why they were desired. Individuals can generally explain the actions, they know that when travelers arrive at their destination, they will want their luggage. The baggage is loaded so that when they arrive it can be unloaded and used.

In general there are four reasons why one might take a certain action within a behavior.

1. to get to the next state in the desired behavior
2. to set up for future actions
3. to thwart events which cause undesirable behavior or hinder desirable behavior

4. for external considerations

Knowledge specification for each of these is considered in the following; appending this knowledge to the representation of behavior links is shown in section 2.3.5.

Normal Causality

For functional behaviors, actions are taken to achieve the desired goal. Usually the sequence of states simply delineates the causal chain of events which eventually leads to the desired result. Each action is performed to set up for the next state, which allows for the next action, etc. This causal chain represents *how* the end result is obtained. It implicitly provides knowledge of *why* each action was performed; the assumed reason for actions in behaviors is to cause the following state, which establishes a causal chain that ultimately results in the desired end state.

No primitives were added to identify the purpose for such behavior links. Unless specified otherwise in the behavior link, an action is taken simply to achieve the next state in a causal chain.

Precondition

Sometimes actions are performed to set up preconditions for future behaviors. One cannot unload the luggage from a plane if it was never loaded. Even though the action of loading is seen early in the overall behavior, it is performed to allow the much later action of retrieving the desired baggage. To represent and identify the purpose for this type of action, a link has a *Rationale* clause. The link specification of *UsingFunction: load.baggage* in the behavior *Prepare.Craft* specifies a *Rationale* that indicates the action was taken so that it can be unloaded upon arrival, i.e., it points to the behavior where the baggage is unloaded.

Contingencies

Often behaviors have contingencies. A certain action is desired unless problems arise, then an alternative action is performed. For example, most automatic garage door openers are equipped with a safety device such that if the door comes in contact with an object before it is totally closed, it will reopen or stop. This prevents damage to cars or items not totally inside or outside. The behavior for the garage door could be represented as in figure 2.4.

Representation for the explanation of why this type of behavior was used includes the use of two primitives. Firstly, there is an additional *Condition* clause so that the contingent behavior can be identified and indicate when the action is used. Because the specified actions are meant to prevent some state, the *Rationale* clause gives more specific detail about the reason this action was taken, i.e., it points to behaviors which occur if the contingency behavior is not taken or is not effective.

Design Considerations

Many behaviors can result in the same final action. Often, however, there are certain constraints which govern the choice of actions used to achieve a function. A particular state transition may be necessary for the achievement of a function, or it may be present to satisfy additional constraints such as optimization. If one is tracing through a behavior to determine if a desired result will occur, it is useful to know when an action is absolutely necessary, or when it is present to maximize safety, economy, etc.

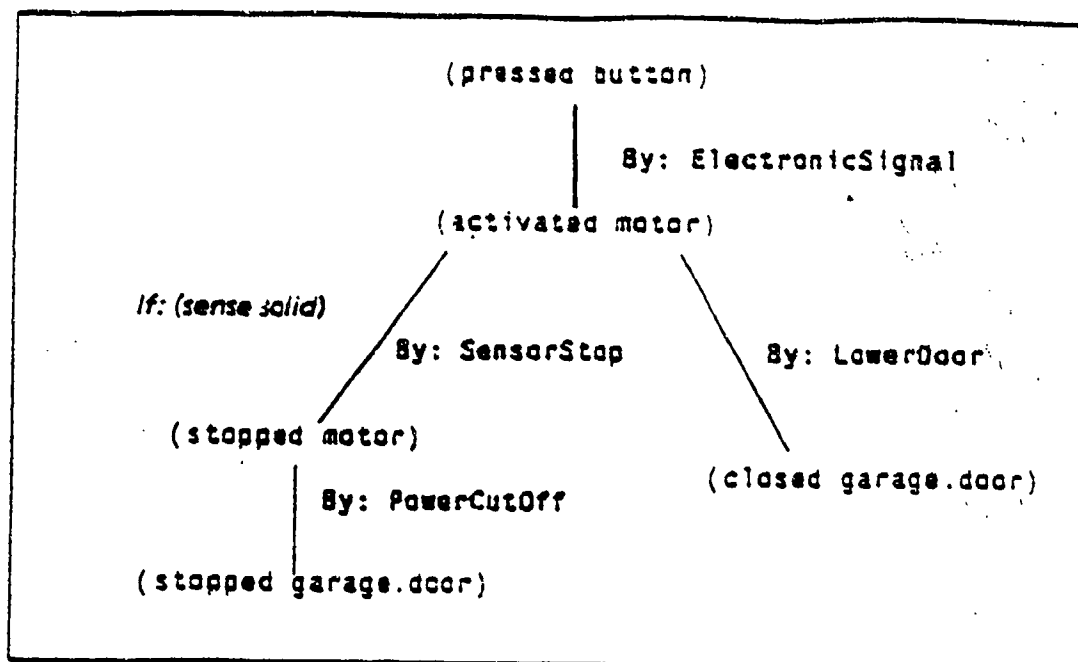


Figure 2.4: Behavior With Contingency

For example, a function of the LiquidFeed System of a chemical processing plant is to supply the liquid reactant. A behavior for supplying the liquid could be to open a valve and simply allow the reactant to flow into the reaction vessel. Instead, the liquid flows past a flow controller and past a temperature controller. The explicit functions of these controllers are to maintain the flowrate and temperature. Their use here is for efficiency reasons; they produce optimum reactant conditions for the reaction. The absence of these functions would still permit the liquid to flow to the reaction vessel, and thus the achievement of the LiquidFeed System's function. The system would have lost optimality but not functionality. The representation of why a design consideration is present, and its significance for achievement of the behavior in which it is present, is achieved through the link primitive DesignContext.

2.3.3 Thresholds

Sometimes components may be only partially working and still be functionally sufficient. The achievement of a task or function can vary in the degree of behavioral completeness depending on the specific context. Some behaviors have BehaviorThresholds wherein the completion of a goal is determined. The threshold stipulation indicates the degree to which any transition must reach completion in order for it to accomplish its part toward the achievement of the final goal.

For example, consider a pump which was designed to produce a pressure of 15 p.s.i., and can now only provide 10 p.s.i. If the behavior in which it was used does not require greater than 10 p.s.i., then for practical purposes, this "malfunctioning" component can still be considered functioning.

The threshold is relative to the change desired, for the particular substance, in the particular device.

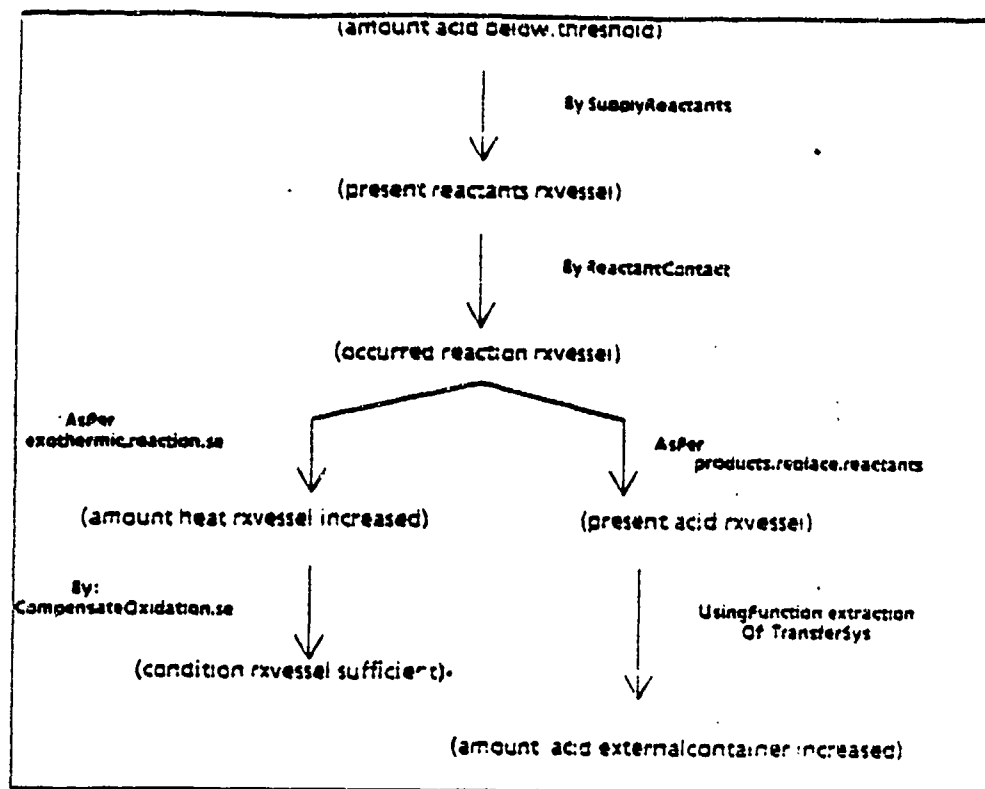


Figure 2.5: Top Level Behavior *oxidation* of a Chemical Processing Plant

with the particular behavior. Thus the information is represented on the link which represents this specific use. Through the use of the primitive BehaviorThresholds on the links, one can specify the degree of completeness necessary and thus determine if the given behavior can accomplish the necessary action to allow the rest of the behavior to continue and be successful in achieving the goal.

2.3.4 Side Effects

A structuring of the device representation around function has provided explicit knowledge of expected functionalities of components. With this model, one can make the distinction between what is the expected function and what are side effects of using the specific behavior for achieving that function. In deep models focusing on physical components, the distinction between which behaviors are due to expected functionalities and which are side effects or potential behaviors is not apparent.

Side effects are an important part of explanation of how a device works. In a chemical processing plant, many of the system's functions are present in order to deal with side effects of the reaction. A major consideration in the plant is that the reaction is exothermic. Thus we see the side effect causal chain described in figures 2.5 and 2.6.

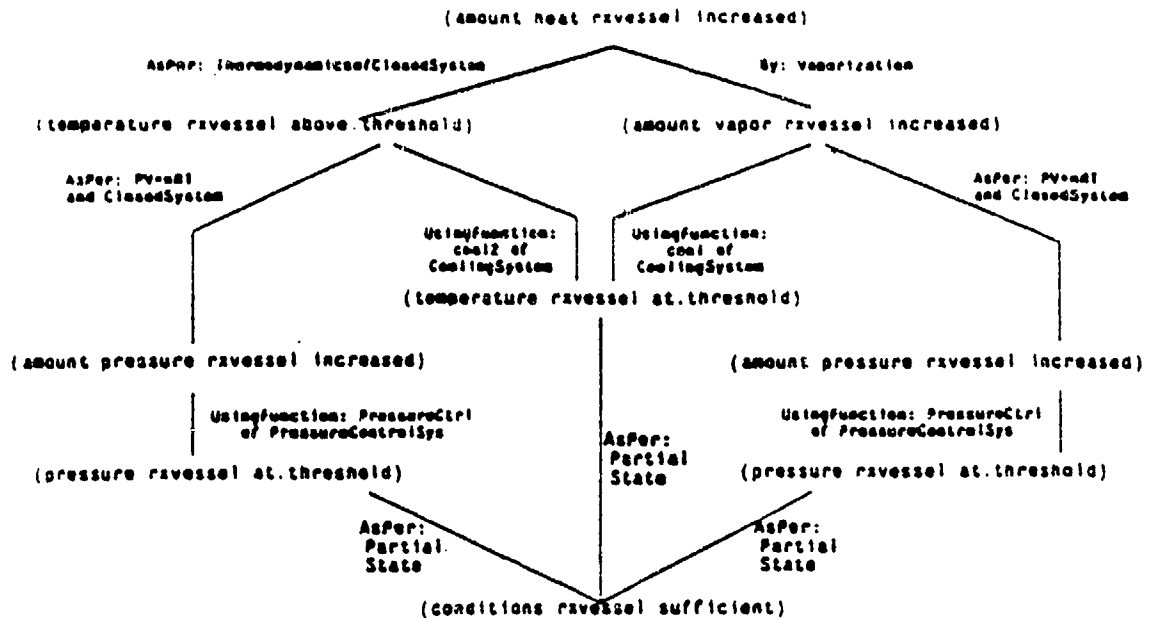


Figure 2.6: Side Effect of Exothermic Reaction

The behaviors of important side effects can be explicitly represented using the behavior primitives and causal chains already mentioned. The distinction that this behavior is a side effect is represented through the link primitive **SideEffect** and is noted on the side effect behavior name by the suffix **“se”**.

2.3.5 Summary of Behavioral Enhancements

A summary of this section on behavioral enhancements is provided through a listing of the link primitives. Each link in a behavior causal chain must have a **LinkType** specified, the remaining primitives are specified depending on why the link is present in the behavior.

LinkType: one of these types must be specified:

- **UsingFunction:** { function } of { device }
- **By:** { behavior }
- **AsPer:** { general knowledge }

Condition: states the contingency condition under which the behavioral sequence will occur

Rationale: a specification of Rationale indicates one of the following:

- When used with a contingency, i.e., if there is a condition specified for this link, **Rationale** points to the behavior that occurs if the contingency behavior (the rest of behavioral sequence after this link) is not accomplished.

- If no condition is specified but there is a specification of Rationale, then this link's action was used in the behavior as a precondition. Rationale points to the behavior for which this link was needed to provide proper conditions.

DesignContext: to identify external considerations for which the action of this link was used, e.g., safety, economy

BehaviorThresholds: to determine the degree of completeness required of this action in order to be within limits for achievement of the behavior

SideEffect: indicates whether the link begins a side effect sequence

The use of these primitives for causal explanation is discussed in the last sections.

2.4 Levels of Understanding: Multiple Viewpoints

The background and area of expertise of an individual plays an integral role in his interpretation of the world. For the task of explaining the mechanisms of a device, the vocabulary, explanatory power and depth of understanding of a mechanic will differ from that of a technician, which will differ from that of an engineer.

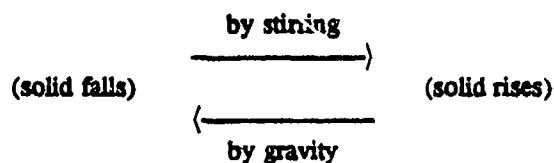
One major source of variance in the knowledge of different individuals arises from one's ability to abstract. This entails the skill and capability to switch between contexts and conceptual levels. In the following sections, I propose a representation which allows access to multiple perspectives for states and behaviors.

2.4.1 Representation of State: State Abstraction

The representation of behavior in the functional representation is based on the use of state transitions. This thesis does not address the specification of a language of state in general, but I would like to address the various levels of complexity which a representation of state must consider. For example, consider the differences between what it means to "be in" the following: a state of confusion, a state of poverty, a state of oscillation, a state of the liquid in the glass having a temperature of 24° Centigrade. This section deals with one form of complexity of state I term "state abstraction".⁵

Process States

Suppose there exists a solid/liquid mixture in which action is being taken to keep the solid from the bottom of the container. One might witness the following causal loop:



⁵ Much of this section is taken from [8]. The author would like to give Dean Allemang special thanks for his efforts.

Here an observer could follow the loop any number of times, but somewhere one takes a conceptual jump and identifies the dynamic process (solid fall, solid rise, solid fall, solid rise...) as a state at a different behavioral level, i.e., identification of the process state (solid suspended). In doing so, he is identifying a new phenomenon; he is packaging a process and seeing it from a higher conceptual viewpoint. These types of conceptual transitions are commonplace in behavior — especially in cyclic or repeated behaviors. Nevertheless, past methods of behavior abstraction do not explicitly address the representation of such phenomenon.

Past Efforts: Behavior Abstraction

Most efforts in the past dealing with abstraction of behavior have adopted the view of abstraction in terms of availability of detail. Two examples, both dealing with detail but in slightly different ways, can be seen in the works of Rieger and Grinberg [12], and Sembugamoorthy and Chandrasekaran [14].

The "mechanisms abstraction" of Rieger and Grinberg accomplishes behavior abstraction by suppression of detail. The idea is to simply remove any state information in the description of the device which is not deemed relevant to the existing situation. For example, consider the following:

state A causes state B
state B causes state C

Using a "defocusing" rule, the system derives,

state A causes state C

Thus information which is not directly pertinent, in this case knowledge of state B, becomes implicit within some causal relationship.

In the work of Sembugamoorthy and Chandrasekaran the behavioral representation provides capability for a hierarchy of detail. Depending upon the level of detail one desires, this model provides access to consecutive levels of a behavior hierarchy through use of the By construct.⁶

Although the results of these techniques vary, the view of abstraction is the same. Given any two states there are potentially an infinite number of states between, so one must always take a stance as to which levels are important. The idea is that details at a low level, which are not relevant to the model's uses, can be suppressed. The representations discussed interpret behavior abstraction as this loss of detail. With this alone, understanding remains limited by the availability of only one viewpoint of behavior. Often an understanding of how something works also requires making transitions between varying conceptual levels. Behavioral representations need to offer the necessary primitives to provide such capabilities. In this section I identify a different type of behavior abstraction from the detail suppression seen above and call it "state abstraction".

Abstraction of State: States as Packaged Behaviors

To be concrete, I will use the example of de Kleer and Brown [4] and Sembugamoorthy and Chandrasekaran [14] of the household buzzer. See figure 1.1. The output we expect from a buzzer is to hear

⁶ In ABEL [10], Patil also allows for causal link abstraction in order to "eliminate intermediate states from causal chains of reasoning." Patil uses a "link aggregation hierarchy" and achieves the same sense of detail suppression.

a buzzing sound. If one follows the causal chain of its behavior, the following is seen: When the manual switch is pressed, T1 and T2 become electrically connected. This allows a voltage to be applied to the coil. Since the coil is magnetic, space1 is now magnetized. Specifically, this makes the clapper area magnetized which pulls the clapper up. Now the closed circuit is broken; T7 and T8 are no longer electrically connected. This causes the magnetic field to be broken and a spring allows the clapper to fall. With this, the circuit is closed again, so the sequence repeats.

This causal chain is, of course, an important part in the explanation of how the buzzer works. Nonetheless, the sequence fails to indicate what is making the buzzer buzz. It is not when the clapper is up, nor when it is down that makes the sound. It is the oscillating motion, the repeated action of the sequence up and down, that causes the effect desired. The clapper is in a dynamic state of oscillatory motion. This causes a buzz. There are actually two conceptual moves here. First, to look at the repeatedness of states as a package which produces a synergistic uniform action (oscillation), and secondly to identify that this action, in a new context, is identified with a desired state — the buzzer is buzzing.

In a simple (no cycle) causal chain, it is often sufficient to understand the device in terms of constituent states resulting in a final state. But for a circular causal chain, it is necessary to recognize that the loop can collectively amount to a distinct device state. State abstraction is proposed as a representational primitive in the sense that it is not built simply from lower level, more detailed states, but it also provides the capability to interpret the cyclic behavior as a dynamic behavioral unit. One needs this capability to identify and explain such concepts, to understand behavior at varying levels of abstraction, and to make the transitions between these levels.

Identification of Cyclic Behaviors

The necessity for using state abstraction especially appears when one tries to represent complex devices, most notably when representing devices in which the behavior has some kind of loop. Through an examination of the literature on devices with loops, I discuss what kinds of loops are interesting to understand, and what sorts of questions about them can be answered.

de Kleer [5] discusses a method for determining the steady output of devices such as a homeostatic control. In such a device, deviations from the steady output are detected, and this information is fed back to some control system, which alters the output in the direction to correct it. This is an example of an **information feedback loop**, in which information about some system variable is processed, and the result is used to affect that same variable. Conceptually there are three main kinds of outcomes that can result from such loops. The behavior of the system can tend toward a constant, as happens in a homeostatic control. Alternatively, the system can oscillate among states, as in a drinking bird toy. Finally, the system may be self supporting, so that the magnitudes of the system variables increase without bound. This happens at a concert hall when the sound system suffers from 'feedback' problems, or during inflationary times.⁷

Weld [19] has examined loops in which a process is repeated until some condition is satisfied. An example of such a system is a frog in a well, who is able to jump three feet up, but then slides back two feet. He jumps until he gets out of the well. This is an example of a **simple process loop**, in which the behavior of the system is understood primarily as the repetition of some process. In addition to the loops with which Weld deals, there are also process loops in which the termination condition is not important, but rather some feature of the system is maintained by the loop process itself. An example of this is stirring

⁷One might recognize these behaviors as essentially three types of oscillation: stable (damped), harmonic, and run-away.

milk in a saucepan to keep the temperature uniform (to prevent the milk from burning). The termination of the loop is not important (we don't care how many times we have to stir the milk while heating it, or where the spoon is in its trajectory when we take the milk off the heat), but rather the continuity of the process is the driving force (we might care how fast we are stirring).

For certain process loops, Weld can determine the values of all state variables in a system when the termination condition is reached, without having to simulate each step of the process separately. de Kleer is able to predict the behavior of some information feedback loops. The purpose here is not to determine the final behavior, but to determine what is needed to understand such systems. In all of these loops, it is important to abstract the fact that a loop is occurring in order to understand the system. Weld converts his loops into functions he can analyze to find the final state of a system. de Kleer uses the first derivative to capture the looping nature of homeostatic systems. Mere suppression of detail does not give these methods their power. These methods work because some particulars are known about the loops themselves and the higher level goals the specific loop is trying to achieve, as in homeostasis. Associated with each specific type of loop are some particular reasoning patterns. Once we identify distinctive types of loops, we can index them to access the appropriate inferences for the loop type.

Representing State Abstraction and Cyclic Behaviors

Because cyclic behaviors are so pervasive in real devices, most device representation systems are equipped to represent the causal chains which participate in such cycles. For instance, both in Rieger and in Sembugamoorthy we see circular links. Both of these approaches are based upon state/transition diagrams and admit the possibility of abstractions. It is not the purpose here to work out the technicalities of an underlying detailed causal representation, but to illustrate the utility of including abstraction of state into the abstractions supported by such a representation.

In order to represent an abstracted state, clearly two things are needed.

- A detailed description or "package" (say, as an elaborated state/transition diagram from Rieger or Sembugamoorthy) of the behavior of the system in that state. In the sense that we understand this as a package, we are identifying a new construct.
- A symbol for the abstract state, which can participate in more state/transition diagrams. That is, the symbol will identify the use of the construct in some higher level context.⁸

With this much one can separate the representation of the repeated motion of the buzzer's clapper from the role the buzzer's sound might play in a more inclusive system. In addition to this information about the particular abstract state, the state can be appended with information about the type of loop (or, more generally, package of behavior) which results in this state. So for instance, one could mark it as being the sort of loop Weld deals with to indicate that such an analysis could be used. The knowledge base for the system has information about each of these loops, such as characteristics of the general process and scripts [13] (or script-like objects) describing the behavior. Each type of loop comes equipped with a package of pertinent inferences of which to index. Essentially, recognition of the type of loop thus corresponds to a "mental move" in terms of inferencing capabilities.

⁸ For example, the state (solid suspended)* is annotated with an asterisk. States identified as such have an AbstractionOf primitive to point to the cyclic behavior or "package" that the state represents.

The abstraction thus provides the device model with the ability to support transitions between conceptual levels. It identifies a phenomenon with some package of behavior (both to a specific detailed description and to a general mechanism) and thus allows access to information which may be relevant for processing needs.

2.4.2 Feedback: Behavioral Abstraction

Understanding of behaviors can also vary in perspectives and levels of abstraction. For example, consider feedback. A chemical processing plant makes use of many feedback controllers to stabilize the flow or temperature of liquids and gases [16]. Explaining the output of one of these controllers, one might state that the temperature is oscillating closely around a given point. A representation of such a state, that of oscillation, is possible via state abstraction, as described previously. But consider an explanation of the more inclusive feedback behavior that resulted in the oscillation. A state transition diagram can represent the causal chain of states, which illustrates a *form* of feedback control⁶, but the representations do not explicitly identify or note that feedback is occurring. An agent who understands the device has both the knowledge of what actions are being achieved together with the realization that these actions constitute an example of feedback. Specifically, they have the knowledge that feedback is being performed along with the information and inference potential that this abstraction indicates.

Representation of a Feedback Controller

I will illustrate the representation and use of behavior abstractions through the specification of a feedback controller used in a chemical processing plant to keep the temperature of the reactants consistent. The function is the type **ToMaintain**.

FUNCTION:

ToMaintain: (liquid temperature)
Range: (30–32°C)
If: (present liquid HeatExchanger.Inlet)
By Feedback6
 ContinuousMonitor: (Feedback6 monitor)
 Until: (difference (temp setpoint) established)
 ContinuousAdjustment: (Feedback6 adjust)
NotBehavior(s): PoorQuality, TooHot
ExternalConsideration: economy, product quality, safety (to provide optimum conditions for the reaction)

Behavior Feedback6 points to the specific state transition diagram for this particular feedback device. The behaviors are shown in figure 2.7.

⁶Representation of systems which use feedback has been demonstrated in other work [3, 6].

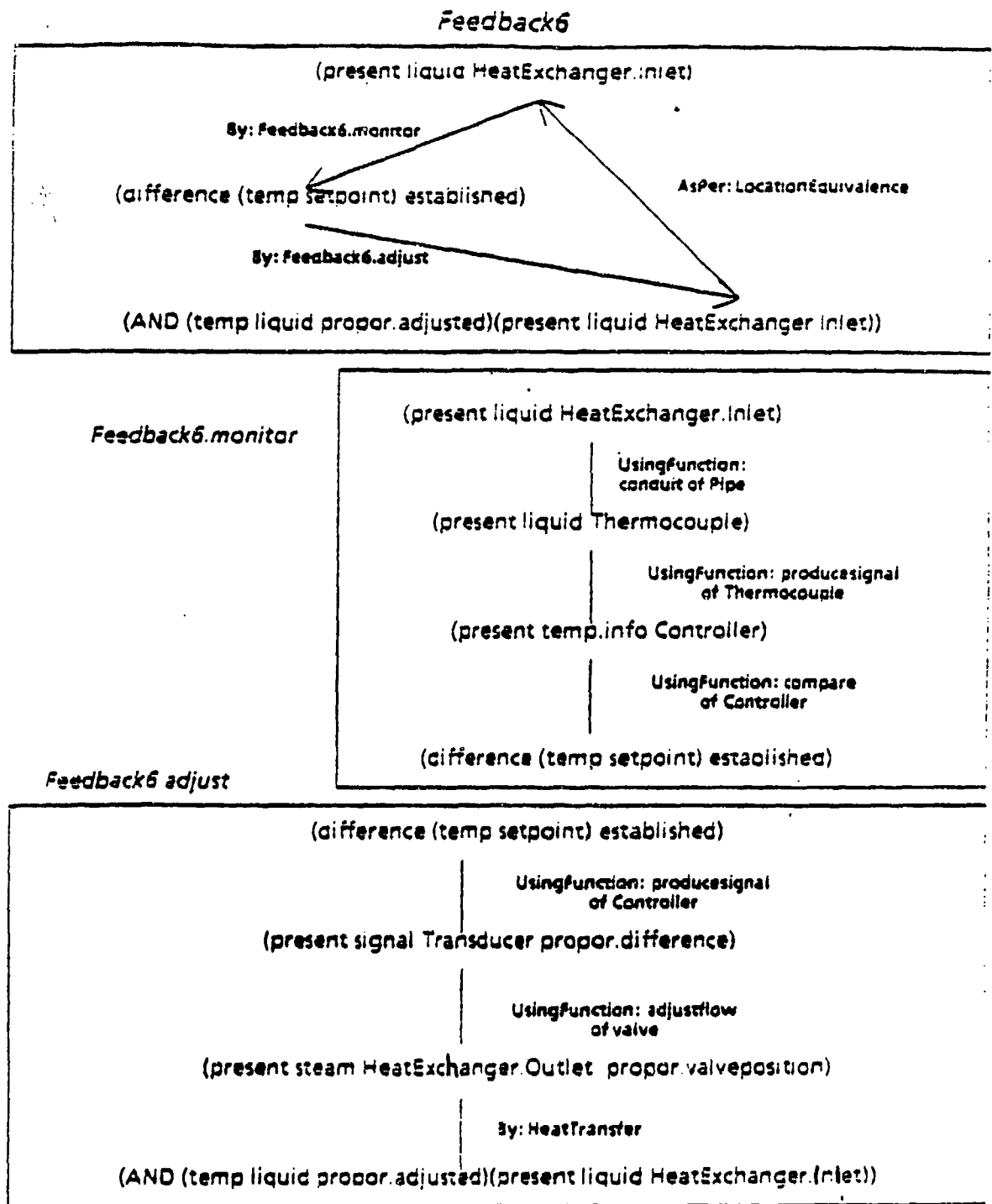


Figure 2.7: Feedback Behaviors for Temperature Maintenance

Behavioral Abstraction: InstanceOf

Because Feedback6 is an instance of a more general behavioral form, its specification points to the higher level abstraction. It is through this information that a transition can be made to the more abstract behavior and the inferences which are applicable in that context can be accessed. Specifically, Feedback6 is represented as follows:

Behavior Object

Name: feedback6

Pointer: (points to behavioral causal chain)

InstanceOf: Automatic Process Controller, Homeostasis, Feedback

Representing Behavior Abstraction Packages

The representation of Feedback6 includes a pointer to the transition diagram which indicates how the goal of maintenance is achieved, together with the identification of the general mechanisms from which this behavior was designed. Each mechanism indicates a distinct behavior type and thus points to the detailed descriptions or packages which provide inference mechanisms and knowledge pertinent to the type. I will illustrate the additional knowledge these packages provide by describing some of the knowledge accessible concerning automatic process controllers.

Automatic process controllers are devices which have the objective of maintaining a controlled variable at set point in spite of disturbances [15]. That is, the objective is to control some outlet variable in order to maintain a desired value.¹⁰

The top level behavior, illustrated in figure 2.8, indicates the three basic operations that must be present for every control system, i.e., measurement, decision, and control. The three operations provide knowledge of necessary components by inspecting the next level of detail, as shown for measurement and decision in figure 2.9.

Information Processing Use

Access to behavior abstractions and general mechanisms is clearly useful in the tasks of redesign, repair or prediction. Here I will focus on the enhancements it provides for explanation.

In addition to the explanation capabilities provided by the knowledge of ToMaintain functions and the explicit causal behavior specified in the Feedback6 loop, one can also answer questions by using knowledge that the loop is a homeostatic feedback loop which is automatically controlled. For example, if the temperature sensor is visibly damaged, and the temperature in the reaction vessel is too high, an operator might wonder how a sensor could cause a change in the temperature. The system could access the information about homeostatic control mechanisms to generate the answer, "The sensor is used in an automated control system. The value reported by the sensor is transmitted to the controller and used to compute the difference

¹⁰Notice that even though the device's main goal is to maintain, it is called a controller. This is because one of the central operations involves controlling in order to achieve the goal of maintenance. The component in this controlling operation is called a controller. I will call the overall system the control system and use the term controller only when referring to this component.

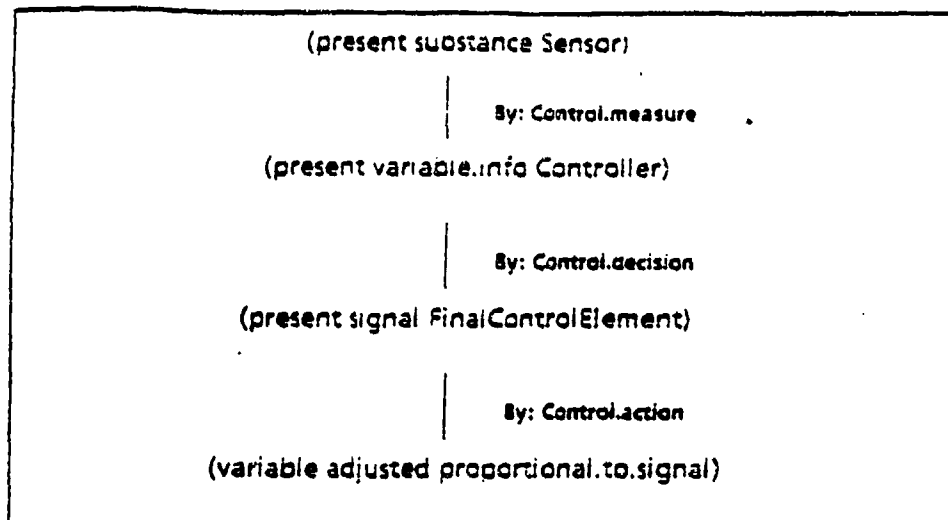


Figure 2.8: Behavior for Control Systems

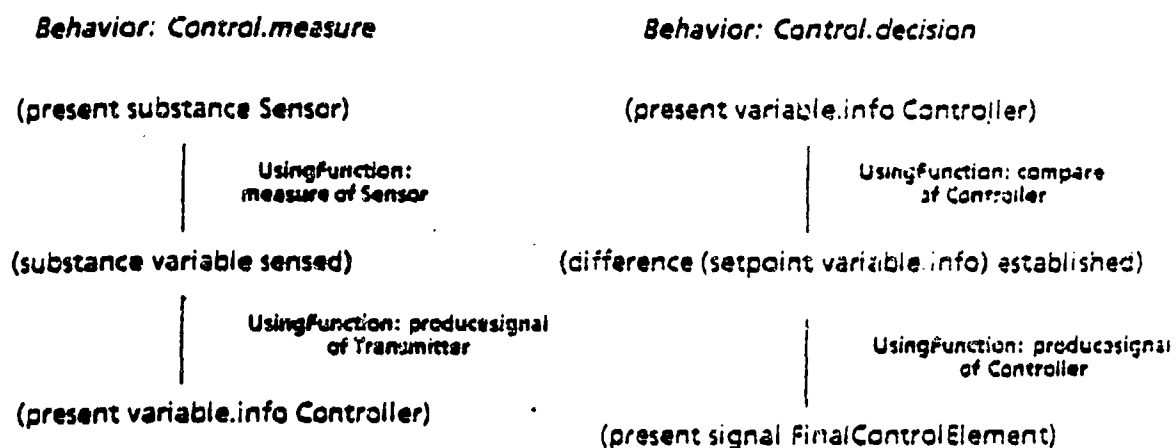


Figure 2.9: Behaviors for measurement and decision

between the actual temperature and the setpoint. The action taken to adjust the temperature is dependent on this difference. An error in either the setpoint or the incoming information with respect to the temperature will make the decision inappropriate. Without a working temperature sensor, the actual temperature will deviate from the setpoint."

For another example, consider the following scenario. Suppose the controller has sent a signal which is outside the range of the actuator, in this case say a valve. The actuator tries to turn the valve too far and the valve gets stuck. The resultant behavior is that the input temperature does not approach the setpoint. Below are explanations derivable first from the Feedback6 behavior and functional knowledge (ToMaintain and ToControl), followed by the extra information available from knowledge supplied by identification of the abstraction of automatic process controller.

Level 1: The Maintenance function in which the actuator is used specifies a Range value which indicates that the given signal is out of range. This implies that the function of the actuator cannot be achieved. Use of the actuator is followed by the Control function of the valve. With the actuator out of range, the control function of the valve will not be operational because the control Relationship is not valid for the value being specified to the valve. Thus the valve cannot produce the appropriate flow. This will result in an incorrect flow of steam which will result in an inappropriate amount of heat transfer. Feedback6 cannot be achieved and a malfunction of the temperature controller results.

Level 2: All control systems have three basic operations: measurement, decision and control action. The actuator is used in the control operation. If this device malfunctions, the controlled variable (temperature) will not be proportionally adjusted as per the decision made by the controller. Because the system is using homeostatic control, a malfunction here means that the variable will not reach the proper limit to maintain the temperature at the desired value.

The specification of abstract schema is useful for explanation. For researchers interested in building models of devices solely to predict behavior at a given level, these abstractions will not be helpful. What is being captured here is not the capability to sequentially follow variables to predict an outcome, but the ability to tell a higher level story. The view is that prediction is not driven solely by constraints due to structure and low level processes, but can be enriched and focused by knowledge of abstract processes and the inferences which they dictate.

Chapter 3

Composing Explanation: Relating Malfunction Hypotheses to Observed Symptoms

In this chapter I discuss the potential of the functional representation as a source of knowledge to be used in problem solving, particularly in the generation of malfunction causal chains which relate malfunction hypotheses to observed symptoms. First, I specify what general assistance a problem solver can expect from the representation, i.e., I describe the kinds of reasoning for which the representation is intrinsically appropriate. I then focus on the problem of the generation of malfunction causal chains which can be used for the justification of diagnostic conclusions.

3.1 General Applicability of a Functional Model

The functional representation provides a package which shows the relationship between structure, function, and the behavior to achieve function. The basic, task-independent, intrinsic capabilities that knowledge of this relationship provides can be described as follows:

Simulation: Given changes in the structure of a device, what can be determined about changes in functionality? This illustrates the "forward" simulation power the representation provides. For example, if a given functional unit is malfunctioning, by tracing the use of its function(s) and considering the effects regarding the achievement of subsequent functions, the overall effect can be determined.

Identification of Structural Cause: Given changes in function (malfunction or reduced effects), what changes in structure could be hypothesized to account for them? That is, given that expected results of a functional component are not present or are affected in some way, one can trace "backwards" (through the steps in the behavior of the function) to see what structural causes could be implicated.

Identification of Functional Components: Given a specific component, what functional purpose does it provide? A packaging via functional units allows the specification of component purpose which

provides an index for identification of use.

The structure of the functional representation, organized around functional packages, provides the focus through which these capabilities can be accomplished. Since, at some level, most problem-solving tasks dealing with devices are concerned with either the achievement of function, or consequences of the failure to achieve function, this description and reasoning power proves useful. The use of the representation in diagnosis seems especially appropriate since diagnosis centers around determining the change in structure that resulted in some malfunction, one of the explicit potentials of the model.

3.1.1 Causal Explanation: The Problem Statement

The problem addressed in this thesis is the following: Given a set of observations and a diagnostic hypothesis, attempt to construct an explanation in the form of a causal story which starts with a diagnostic hypothesis and ends with one or more of the observations to be explained. In this chapter, I examine how the functional representation can be used for this purpose — considering both its capabilities and limitations. Technical definitions of a few terms may be useful.

Observations: observable states, including a subset which are malfunctions of the device subsystems or components. The following distinctions about observations are useful:

- **Symptoms:** abnormal states which are indicative of malfunctions and trigger the diagnostic process, e.g., specification of a drop from normal pressure.
- **Malfunctions:** observations which *are* malfunctions, e.g., specification of a faulty pressure regulator. Malfunction observations are generally also symptoms.
- **Observable states** which provide information about the device but do not directly correspond to abnormalities, e.g., specification of temperature or pressure readings. Typically in a complex system, a large number of observations are used in the diagnostic process which provide focus for the problem-solving but do not necessarily indicate problems (e.g., sensor readings).

Diagnostic Hypotheses: the set of malfunctioning components or missing (but expected) relationships between components. Each in the latter should sooner or later, manifest itself as the malfunction of a subsystem within which the components lie.

Causal Explanation: Normally one expects a diagnosis to causally "explain" the symptoms, even though in general the diagnosis actually should explain all the observations. The explanation provided here takes any given set of observations to be explained and tries to propose a causal path from the diagnostic hypothesis to these observations.

The Explanatory Result

The explanation sought can be formally stated as follows:

$$\text{diagnostic hypothesis} \rightarrow s_1 \dots \rightarrow s_i \dots \rightarrow s_N$$

where each s_i is either (1) an internal state which is causally relevant in producing an observation, but is itself not a malfunction, (2) a component or subsystem malfunction, or (3) an observation at the device-level. The explanation system developed in this work produces explanation chains where the members are limited to the last two, i.e., malfunctions or observations.

Issues to be Addressed

Issues which need to be addressed for the generation of this causal chain include the following:

- generating the causal links from malfunction → malfunction
- generating the causal links from malfunction → symptom¹
- composing the causal story when there is more than one malfunction in the diagnostic hypotheses, especially those which interact

In the following, I provide and explain an algorithm to generate the causal explanation. I will address these issues as they arise during the explanation of the algorithm.

3.2 Generating the Malfunction Causal Chain

3.2.1 The Primary Control Mechanism

In the same way the functional representation provides an organization to allow simulation of how *expected* functionalities are achieved, it can also serve as a backbone to trace the effects of not achieving certain functions — thus identifying potential malfunctions.

The organization of the functional representation gives both forward and backward reasoning capability, i.e., it can trace from the hypothesized malfunction to the observed malfunctions and symptoms (forward), or it can trace from observed malfunctions to the hypothesized malfunction (backward). Because both the observations and the diagnostic hypotheses have been identified once diagnosis is complete, the functional representation could potentially be used to perform either form of control. In this thesis, I provide an algorithm which demonstrates the forward simulation potential.

3.2.2 The Algorithm

If device A is malfunctioning, then devices which use device A (say devices B and C) have a high probability of malfunctioning as well. Similarly, devices which use B and C may malfunction, etc. The malfunction causal chain is achieved through the following algorithm which has been condensed to illustrate main points.

1. Set Observations to the symptoms and malfunctions to be explained. Set MalfunctionList to the hypothesized malfunction set provided by the diagnosis. Initialize MalfunctionObject to an individual malfunction in this set (diagnosed hypotheses and their relationship to observations are considered individually).
2. Find all functions which made use of the function which is malfunctioning (MalfunctionObject), call this set PossibleMalfunctions.
3. For each element in PossibleMalfunctions (call the specific function PossMal) consider the effect of MalfunctionObject on the function.

¹For the sake of clarity, since a malfunction might also be an observation to be explained, I will use the term *symptom* to indicate observations regarding state information, i.e., either states which denote specific abnormalities or simply observed states which have been requested to be explained.

- if no effect on PossMal then remove from PossibleMalfunctions — MalfunctionObject is not causing future problems. Consider the next element in PossibleMalfunctions.
 - else maintain (Malfunction → Malfunction) explanation chain; MalfunctionObject is now known to cause a malfunction to PossMal. Note that this step will ultimately place any potential malfunctions in a malfunction chain, including those which are in the set of Observations. Continue.
4. Check the states in the behavior of the affected PossibleMalfunction. Would noncompletion of these states explain any symptom(s) in Observations?
 - if yes, append to ExplainedSymptoms and print the chain which led to this symptom. Complete the malfunction explanation chain by continuing.
 5. Set MalfunctionObject to PossMal.
 6. Repeat process from step 2 until all symptoms are in ExplainedSymptoms or the top level behavior of the device has been reached.
 7. The Process from step 1 is repeated until all elements MalfunctionList have been considered.

Step 2 is easily accomplished through the component hierarchy of the functional representation and knowledge of SubFunctions. Step 3 and 4 are more intricate and involve both knowledge of functional type and the achievement of behaviors, as illustrated in the following sections.

3.3 Tracing the Algorithm: A Malfunctioning Chemical Processing Plant

I will illustrate the steps of the algorithm by providing the system's explanation of a malfunctioning Coolant System in a chemical processing plant. First, I provide a brief, partial description of the representation of the chemical processing plant.

3.3.1 Representation of a Chemical Processing Plant

References to figures 3.1 – 3.3 will assist the reader in following the causal explanation chains given in this chapter. Figure 3.1 is a schematic diagram of the chemical processing plant that has been functionally represented. The hierarchy in figure 3.2 shows a partial representation of the functional components with their intended functions (functions are specified under component names). The top level function, *produce.acid*,² is achieved by behavior *oxidation* shown in figure 3.3.

3.3.2 The Explanation Problem

Algorithm: Step 1

The Coolant System (identified at the right of figure 3.2) is used to provide coolant water to a Condenser so that it can be used to transfer heat from the vapor in the Condenser. Suppose the coolant water has been

²The acid produced is a solid, terephthalic acid.

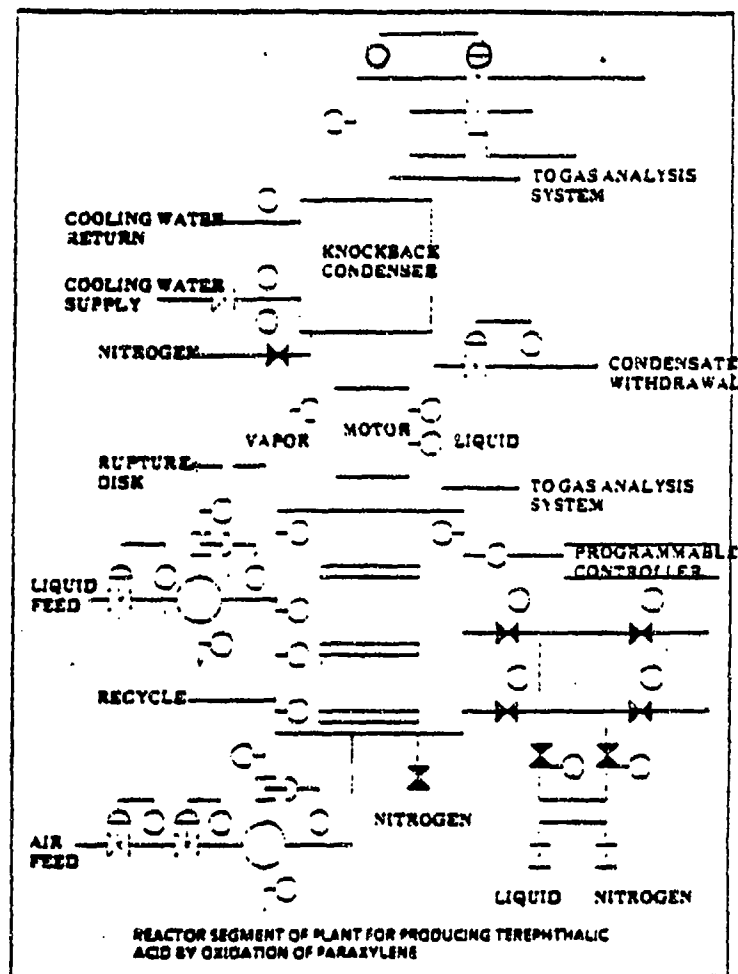


Figure 3.1: Schematic of Chemical Processing Plant

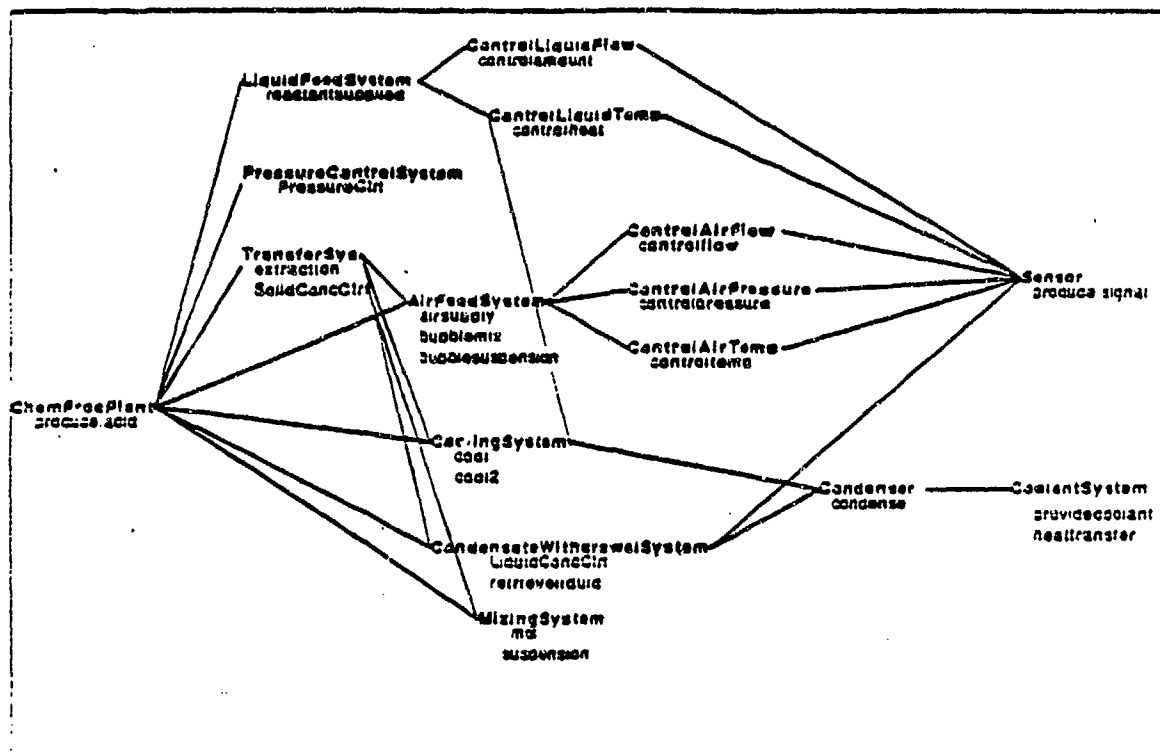


Figure 3.2: Functional Component Hierarchy

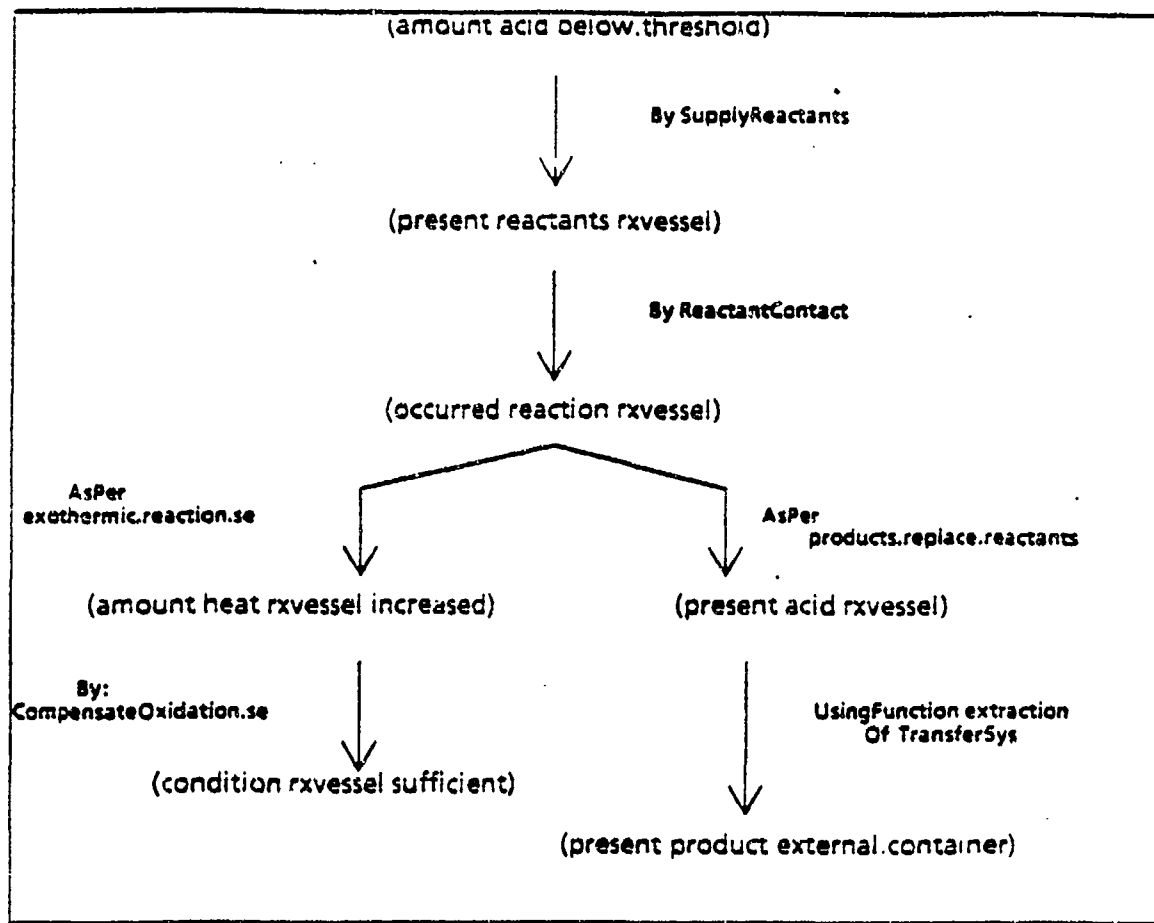


Figure 3.3: Behavior *oxidation* for Function *produce.acid* of a Chemical Processing Plant

completely cut off. A diagnostic system has concluded that a malfunction of the function *provide.coolant* of the Coolant System explains the symptoms of NOT (present product external.container) and NOT (temperature rxvessel at.threshold). Specifically, MalfunctionObject is {*provide.coolant* of Coolant System} and the Observations to be explained are {NOT (present product external.container), NOT (temperature rxvessel at.threshold) }.

Algorithm: Step 2

The system inspects the functions and behaviors of the chemical processing plant to determine where the function *provide.coolant* is used. The search returns the set

{(function HeatRemoval condense)}

which indicates that the function *provide.coolant* is used once, in the behavior Heat Removal of function *condense*. This is the set of PossibleMalfunctions.

Algorithm: Step-3

To determine the effects of the malfunction of *provide.coolant* on the function *condense*, one must consider the possible consequences of malfunctioning components. In general, the malfunction of a component in a device can cause one or more of the following three consequences:

NOT Function: expected results of the function are not present. Given that the function is not producing the expected results within behaviors, what states in those behaviors will not occur, and will lack of this functionality cause the malfunctions of functions in which the malfunctioning component was used?

Parameter Out-of-Range: expected results of the function are affected, but behavior is still achieved to a limited degree. Sometimes components may be considered malfunctioning yet can still achieve the behavior (or value of some substance parameter) needed for future use.

New Behaviors: the malfunction results in behaviors and states which were not those intended for normal functioning.

The type of consequence indicates the effects of the malfunction within behaviors where they are used, and thus potential malfunctions and symptoms. In this device, the Condenser uses the *provide.coolant* function of the Coolant System to provide coolant so that it can be used for the transfer of heat from the vapor, see figure 3.4. If there is not enough coolant or if the temperature of the coolant is not low enough to allow the vapor to reach dewpoint, the desired amount of condensate will not be produced by the Condenser. The identification of the necessary amount of coolant required in order to achieve the desired function of *condense* is specified by the BehaviorThreshold link primitive within the link specification which dictates the use of the function *provide.coolant*.

LinkType: **UsingFunction:** *provide.coolant* of CoolantSystem

Condition: NIL

Rationale: NIL

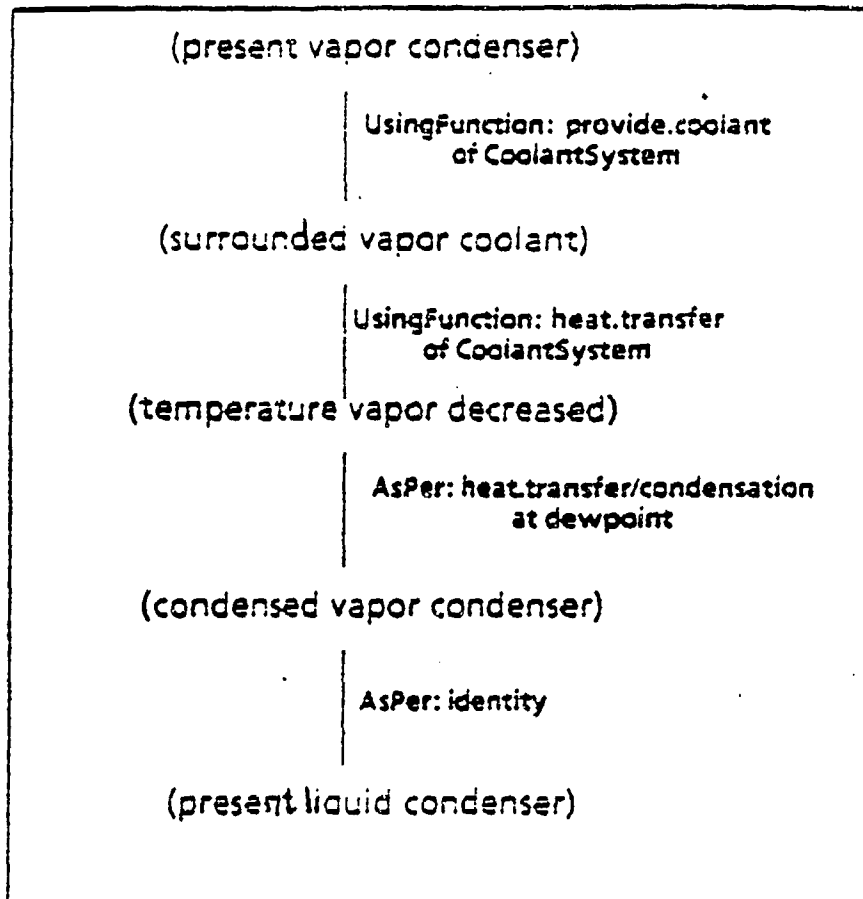


Figure 3.4: Behavior RemoveHeat of Function *Condense*

DesignContext: NIL

BehaviorThresholds: (GREATERP (flowrate coolant condenser) x)

SideEffect: NIL

In the case that has been diagnosed, the coolant water has been completely cut off; the flowrate is not greater than the necessary amount. *Provide.coolant* is "NOT functioning" — the function is not providing expected results. The value of NIL for the link primitives **Condition**, **Rationale** and **DesignContext** indicate that the action was taken for "normal causality", the default reason, as discussed in section 2.3.2. This means the action described by the link was used in order to achieve the next state and ultimately the desired end state (present liquid condenser). The Condenser will consequently malfunction because the necessary states are not being achieved. In this situation, the **MalfunctionExplanationChain** will become *provide.coolant* → *condense*.

Alternatively, the Coolant System could be malfunctioning but still cooling just enough so that a satisfactory amount of condensate is produced — "Parameter Out-of-Range". Here the **BehaviorThreshold** is achieved, the Condenser would not be considered malfunctioning and thus not viewed as the cause of further problems in the device. This knowledge is used to prune the search of potential causal chains. The explanation system would print the explanation chain thus far, in this case it would be trivial. It would then consider the next element in **PossibleMalfunctions** to start a new causal chain.

The function *provide.coolant* was needed to provide a certain threshold value. It did so in this case within the behavior *RemoveHeat* of *condense*.
Effects were observed only in the following:

NOT *provide.coolant* causes NIL
i.e., no further problems in this causal path.

Algorithm: Step 4

Given the situation that *provide.coolant* is causing the *condense* function to malfunction, the explanation system now considers whether this malfunction explains any of the symptoms. The assumption is that, if the action defined by the link was taken to achieve "normal causality", the remainder of the behavioral causal chain is no longer valid and the states which follow the use of the malfunctioning component in the behavior will no longer be achieved. The explanation system compares the symptoms, in the set **Observations**, with the states established after the use of the function *provide.coolant* in the behavior *RemoveHeat*. This includes states accessed through the behavioral hierarchies which provide the detail on how the higher level states occurred. At this point, no symptoms are explained.

Notice that this technique for identifying symptoms limits an explanation system using the functionally represented device to only explaining symptoms of the form NOT(expected state).

Algorithm: Steps 5 and 6

Given that no symptoms have been explained, **MalfunctionObject** is now set to the function *condense* and the process is repeated. Step 2 searches to determine where the function *condense* is used. This time the

search returns the set

```
{(function KnockBack retrieveLiquid)
  (function RemoveHeat cool)
  (subfunction KnockBack retrieveLiquid MixtureLevelCtrl)
  (subfunction KnockBack retrieveLiquid LiquidLevelCtrl) }
```

which indicates that the function *condense* is used in the functions *retrieveLiquid* and *cool*, and also that *retrieveLiquid* is a *SubFunctionOf* the functions *MixtureLevelCtrl* and *LiquidLevelCtrl*.

Recall that subfunctions are functions which may not be found in a trace of the main behavioral causal chain of a device's function but are specified because they provide preconditions or *Provideds* for primary functions. Thus a malfunction in *retrieveLiquid* directly indicates malfunctions in *MixtureLevelCtrl* and *LiquidLevelCtrl*.

Simulation Focus Due to Link Specifications

Thus far the algorithm has used concepts of *BehaviorThreshold*, "normal causality", and *SubFunctionOf* to determine consequences of a malfunction. Identification of *why* actions are taken within a behavior is also used to focus the trace for further malfunctioning components and symptoms, i.e., to consider effects of malfunctions (algorithm: step 3). The roles of links, i.e., normal causality, design considerations, contingencies, preconditions, and side effects, were discussed in section 2.3. Given that these alternate pathways are available for simulation and the generation of the malfunction causal chain, rather than an indepth trace of every step of the algorithm for this particular problem, the following section provides the final outputs (elaborated causal chains from diagnosed malfunction to observations) produced by the explanation system for this problem.

3.3.3 Final Output: The Causal Explanation

The system produces the following three casual stories.

Causal Story 1: Generation of Causal Connections

As shown in figure 3.5, the behavior *SupplyReactants* uses the functions *retrieveLiquid* and *LiquidConcCtrl*. The link specification for the *retrieveLiquid* function indicates that this action in this behavior is used as a design consideration in order to recycle the liquid reactants, i.e., it has the link primitive:

DesignContext: (constraint economy (conserve reactant.liquid)).

If this economic measure is not necessary, then loss of the function will not cause problems in that behavior.³ Given that the design constraint is pertinent, the system prints the following:

The symptom NOT (present product external.container) is
is explained by the following chain:

NOT provide.coolant causes
malfunction in condense which causes

³ Similar to *BehaviorThresholds*, this knowledge serves to prune the search space of potential causal links.

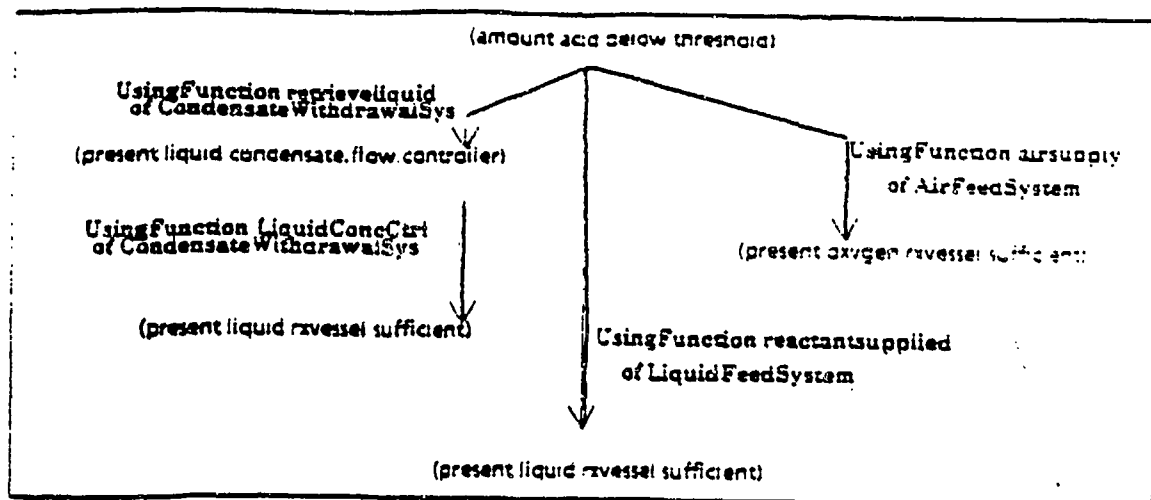


Figure 3.5: Behavior for SupplyReactants

malfunction in retrieveLiquid which causes
malfunction in LiquidConcCtrl which causes
problems in behavior SupplyReactants
which is used in behavior oxidation and indicates
malfunction of the top level function and results in
NOT (present product external.container)

The following symptoms are not explained:
NOT (temperature rxvessel at.threshold)

The idea here is that if the required amount of reactants is not available, the product is not produced as desired and thus can not be retrieved. The explanation system generates this chain by using the following information: *Provide.coolant* caused a malfunction in *condense* because it did not achieve the **BehaviorThreshold** needed within *condense's* behavior. A malfunction in *condense* caused a malfunction in *retrieveLiquid* because its achievement was required (normal causality) to attain the desired behavior for *retrieveLiquid*. *RetrieveLiquid* caused a malfunction in *LiquidConcCtrl* because it was needed to provide the preconditions (**SubFunctionOf**) for *LiquidConcCtrl* and it preceded the use of *LiquidConcCtrl* in the behavior *SupplyReactants*, figure 3.5. Although *retrieveLiquid* and *LiquidConcCtrl* were only present for economic reasons, they were considered necessary and thus their failure effected the behavior *SupplyReactants*. *SupplyReactants* was used in the behavior *Oxidation*, figure 3.3, to achieve the state (present reactants rxvessel). This state was necessary for the completion of the behavior and thus non-achievement here denotes non-achievement of further states in the behavior, particularly NOT (present product external.container).

Causal Story 2: The Use of Side Effect Inspection

The system continues and finds a causal connection for the second symptom, NOT (temperature rxvessel at.threshold).

The symptom NOT (temperature rxvessel at.threshold)
is explained by the following chain:

NOT provide.coolant causes
malfunction in condense which causes
problems in behavior removeheat
of function cool

Since *cool* is not a top level function of the chemical processing plant, the trace continues until all consequences are determined.

The symptom NOT (temperature rxvessel at.threshold)
is explained by the following chain:

NOT provide.coolant causes
malfunction in condense which causes
malfunction in cool which causes
problems in behavior compensate.oxidation.se
a notable side effect behavior used in
oxidation and indicates
NOT (temperature rxvessel at.threshold)

The following symptoms are not explained
(present product external.container)

Notice that this explanation identifies that the symptom was observed in a side effect behavior (compensation for effects of the reaction) rather than a behavior of the main functionality (production of acid).

The statement of which symptoms are not explained indicates those that were not explained in the specific causal chain. A final statement is made when the system has inspected all pertinent causal chains.

Causal Story 3: Using Subfunction Connections for Causal Focus

The final causal path is achieved via causal connections obtained specifically through the knowledge of subfunctions. The function *extraction* has a provided clause which specifies that the solid acid slurry must have the proper consistency so that flow through the extraction tube is possible. The function *SolidConcCtrl* is present in this device for the sole purpose of producing these conditions for *extraction*.

Figure 3.6 shows how *SolidConcCtrl* is achieved. Its purpose is to keep the solid suspended and the concentration in the reaction vessel at the proper consistency. In the *CondensateWithdrawalSystem*, the *retrieveliquid* function uses the *Condenser* to retrieve the condensate from the vapor produced. The *MixtureLevelCtrl* function then uses a feedback controller to maintain the flow and thus the desired amount of

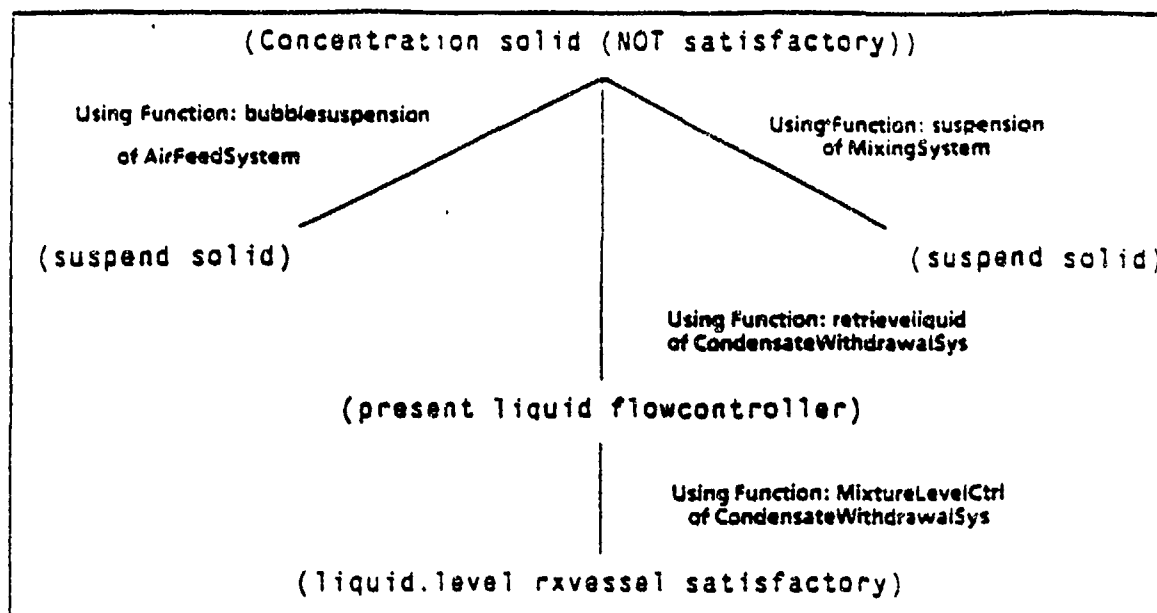


Figure 3.6: Behavior for SolidConcCtrl

liquid in the reaction vessel — which ensures that the acid slurry has the proper consistency. If the liquid is not retrievable, then obviously the condensate flow cannot be controlled and consistency of the acid in the vessel is not maintained. The explanation system provides this explanatory story as follows:

One function affected by provide.coolant is SolidConcCtrl which is a necessary subfunction of extraction

The symptom NOT (present product external.container) is explained by the following chain:

NOT provide.coolant causes
malfunction in condense which causes
malfunction in retrievaliquid which causes
malfunction in MixtureLevelCtrl which causes
malfunction in SolidConcCtrl which causes
malfunction in extraction which causes
malfunction in produce.acid which causes
NOT (present product external.container)

All symptoms have been explained.

3.3.4 Implementation, Enhancements, and Potentials

The explanations given thus far were generated by an explanation system using the functional representation for causal focus. In the current implementation of the explanation system, only a portion of the functional representation's explanatory potentials have been exercised. Some of the enhancements described in Chapter 2, particularly the use of function types and state and behavior abstraction, were not used in this version. In the next sections I address how these additions to the functional representation could be used to further enhance the explanation capabilities.

3.4 Additional Potential Uses of the Functional Representation

Thus far the explanatory system considered only the non-achievement of a function's behavior to identify potential malfunctions. However, the determination of achievement of a function (or identification of malfunction) can involve more than tracing the function's behavioral causal sequence. The specification of a particular function type provides knowledge of functional expectations and can thus indicate inferences relating to that functional form. Specifically, each distinct type of function provides knowledge of what it means to achieve, and thus not achieve, that type of function.

3.4.1 Using Knowledge of Function Type to Determine Malfunctions

Consider a function of type *ToControl*. The concept of control, and thus the function specification, indicates that two important conditions must be valid: (1) there exists the power to control, and (2) the control relationship, which specifies how the controlled event changes as a function of the controlling event, is in fact present. Specifically, a water faucet might work in that it allows water to flow, but the device does not have control without the above two traits. The determination of whether a component which is a *ToControl* function will malfunction should address these aspects.

Power to Regulate

Specification of *ToControl* functions provides knowledge of what is being controlled and how. As described in section 2.1.4, the faucet control function fails if the controller has no power, i.e., if the valve becomes stuck, the handwheel is no longer adjustable, or there is no water to regulate. Such malfunctions might still allow water to flow, but not at the desired amount. Effects of malfunction due to loss of this control aspect are observed through *NotBehaviors*.

For example, a feedback mechanism is used in the chemical processing plant to maintain the flow of the liquid reactants into the vessel. If the signal, which specifies how much the flow should be adjusted, is out of range, then the valve sometimes gets stuck and later adjustment signals are ignored. Rather than the flow being maintained at the set threshold, it remains constant. Using the simulation capabilities of the functional representation, this constant flow can be traced to the actuator (in this case, the faucet handwheel) in the *controlamount* function of *ControlLiquidFlow*. A forward causal trace of the feedback mechanism used to perform *controlamount* shows that the controller is sending the proper information to the actuator for adjustment, but the actuator is not responding by producing the proper flow. At this point, the control aspects of the *ToControl* function can be used to complete explanation of this malfunction. Specifically, the faucet is not properly controlling because it no longer has the power to regulate. The *NotBehavior*,

Handwheel.stuck, indicates that a constant flow (relative to the position of the handwheel) is now expected no matter what signals are sent to adjust the flow.

Relationship Validity

The second aspect of control is that a relationship exists between the controller and controllee. Control functions will malfunction if this relationship is no longer valid. The proper functioning of a faucet does not mean simply that water is flowing, but that it is flowing at the proper rate given the handwheel position. Specifically, for a ToControl device, it is not simply a matter of producing any output, but the output must be directly related to the input (via the Relationship) or the component is malfunctioning.

3.5 Intrinsic Limitations of the Functional Representation for Explanation

The intrinsic limitations of the functional representation for explanation arise from its intrinsic limitations for simulation. The representation uses prepackaged behaviors which are organized around the expected functions of a device. Simulations of malfunctioning devices are thus limited to statements of what expectations are "not" occurring. For example, in the current implementation of the explanation system, the only observations that can be directly explained by the functional representation are malfunctions which are determined due to NOT achieving the behavior which accomplishes the function, and states which are in expected behaviors but are NOT achieved.

This limitation effects the capabilities for explanation in two significant ways. First, the functional representation is not capable of generating causal stories of malfunctions which interact unless the device representation has this interaction explicitly represented. Similar problems regarding the interactions of malfunctions arise in diagnosis [17]. Secondly, "new" behaviors, i.e., behaviors which are not those intended for normal functioning but which arise due to a change in device structure, could potentially lead to symptoms which cannot be explained using the functional representation.

3.5.1 Malfunctions Causing New Behaviors: Matching Symptoms

Consider a device which has a malfunctioning amplifier. The functional representation can be used to conclude that the device will NOT amplify, but it cannot be used to determine that the device is now, in fact, oscillating. Typically, there are three modalities by which the effects of new behaviors can be determined.

1. The use of other forms of simulation, i.e., from structure to behavior, can be used to generate behaviors (e.g., spatial, topological, temporal). Here the reasoner must make sure that the structural description has enough knowledge, in principle, to produce the required result. Current qualitative simulation techniques are limited to identifying behaviors at the same level of abstraction as components. The capability to make transitions in abstraction levels, for example, from "current" and "voltage" to "oscillation", is an open-ended issue.
2. Salient malfunction behaviors can be explicitly modeled as a part of the device representation. In medicine, especially diagnosis, examples of this are prevalent. Here the physician wants a mechanistic model of some phenomenon — but the phenomenon of interest is a disease. Thus his "working" model

is a functional model of the body based on malfunctions (disease) of the body rather than the expected functions of a healthy body. These are the causal chains in which the physician is interested and has compiled. The next section illustrates how the specification of functional types provides access to precompiled malfunction information through the functional representation.

3. The use of generic classes of behavior, as in general, background knowledge (e.g., knowledge of liquid flow) can be used to determine behavior.⁴ The idea is that reasoners have knowledge of general situations from which they can infer expectations for specific situations.

An explanation system could use the enhanced functional representation to identify observations obtained through precompiled malfunction chains or generic classes of behavior, as discussed in the following sections.

3.5.2 The Functional Representation, Precompiled Malfunction Behaviors, and Not-Behaviors

The specification of functional types **ToMaintain**, **ToPrevent**, and **ToControl** imply that they are used in a more inclusive behavior. Explanations of *why* they are used are often given in terms of *what* behaviors occur in the absence of the function. The functional representation allows access to such precompiled malfunction behaviors, and symptoms described therein, through the **NotBehavior** primitive in functional specifications. Such knowledge provides the functional representation with the ability to explain not only what functions and expected states will not be achieved, but also certain well-known symptoms of malfunctions.

Consider the rupture disk described in section 2.1.3. The rupture disk is used **ToPrevent** the chemical processing plant from blowing up due to excessive pressure.

FUNCTION: reduce.pressure of RuptureDisk

ToPrevent: (reactionvessel blowup)

Threshold: (tensilestrength value)

PassiveProperties: solid, impermeable until pressure=tensilestrength

If: (rupturedisk present)

When: (pressure reactionvessel at.threshold)

By: behavior blowout

Provided: assumptions for RuptureDisk behavior

NotBehavior(s): behavior NotBlowup

The **By** primitive for **ToPrevent** functions points to the behavior sequence which represents the behavior of the component itself, i.e., how the rupture disk reacts. The **NotBehavior** specifies context behavior to illustrate the purpose of the device within the overall system. This behavior for the rupture disk, **NotBlowup**, is shown in figure 3.7. Such knowledge allows explanation systems to inspect behaviors of functions used specifically to prevent certain states, and also important side effects due to these actions.

This knowledge could be used, not only to explain how certain states observed in the **NotBehavior** were caused, but also to explain further malfunctions. For example, the causal chain explaining how use of the rupture disk might cause the symptom **NOT** (present product external.container) can be generated since the

⁴ Much the same idea as Schank's MOPS [13].

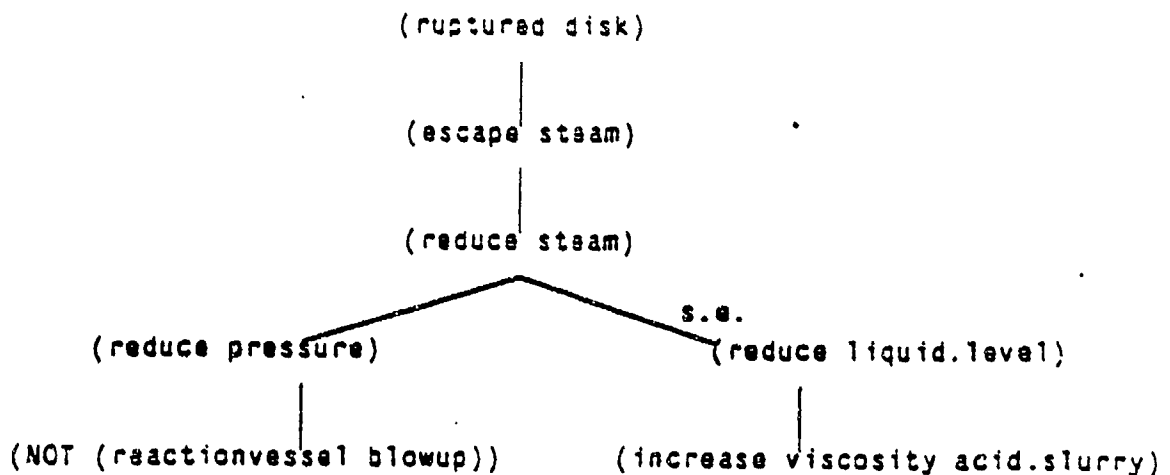


Figure 3.7: Behavior NotBlowup

extraction function has a provided clause which specifies a certain consistency for the acid.slurry and the side effect behavior shows that a ruptured disk causes an increase in the concentration.

For another example, if the ToMaintain function *SolidConcCtrl* (figure 3.6) malfunctions because it is not maintaining the state of (solid suspended), the behavior in figure 3.8 is expected. The behavior indicates problems both for the *AirFeedSystem*'s functions and in the *extraction* function of the *TransferSystem*: knowledge which could, in a system which utilizes more knowledge of state, further expand the causal malfunction trace.

3.5.3 The Functional Representation, Generic Classes of Behavior, and Abstractions

The third method to determine "new" behaviors was to use knowledge of generic classes of behavior. For instance, one expects the same consequences from a leaky faucet in the kitchen, as a leaky faucet in the bathroom, as a leaky faucet on a park's drinking fountain. In fact, one has a concept of "leak" in general. Access to these knowledge constructs could assist in making common causal connections, such as a leak in a device could cause a puddle underneath the device. Specifically, a state of (faucet leaks) can be causally connected to (floor wet).

The functional representation does not explicitly represent the knowledge of all concepts used within states; however, the use of state and behavior abstractions, as discussed in section 2.4, does provide a means to represent some of these abstractions and access to the inference mechanisms which are commonly used concerning these concepts. For example, the provided clause of the function *conduit* of a pipe may specify that the function's behavior works Provided: NOT (pipe plugged). Upon malfunction, this state could trigger an inspection of what it means to be plugged, along with knowledge of what happens when things are plugged, i.e., things back up, etc., and this knowledge could be potentially linked to symptoms.

Similarly, suppose the coolant water in the chemical processing plant is cut off and a symptom is described as "the product extracted is too thick". General knowledge of what makes a slurry too thick can provide

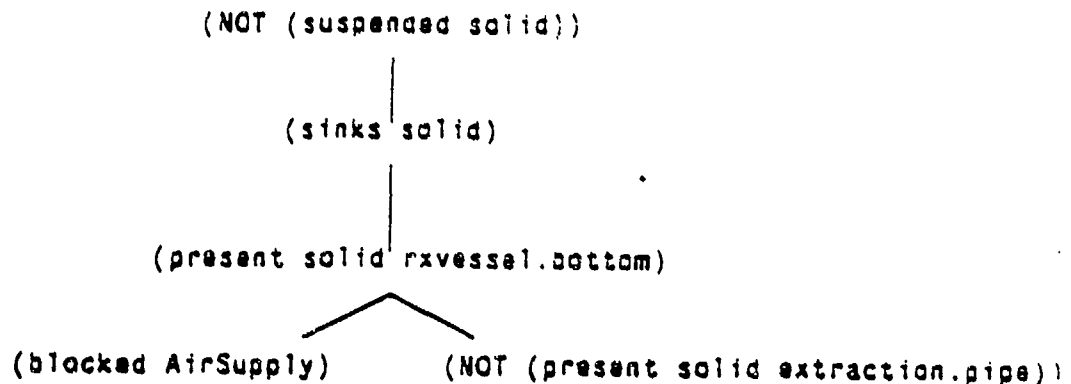


Figure 3.8: Behavior solid.sinks

information to help the system complete its causal trace. That is, general knowledge indicates that a liquid may be "too thick" either because there is too much solid or not enough liquid. The explanation system could then make a connection to the causal chain already produced using the functional representation, i.e., the chain linking the absence of coolant water to a reduction of liquid in the reaction vessel.

3.6 Summary

The functional representation is effective for the generation of causal explanations of diagnostic conclusions because of its organization and focusing potential. Using the functional representation, the number of causal sequences which might directly explain symptoms is pruned to the behaviors of functions which will potentially malfunction. In addition, the representation can be used to direct an explanatory system to relevant constructs. The examples shown in this chapter illustrate the most direct uses of the functional representation, its primitives and organization, for the task of malfunction causal explanation. Use of the representation as a backbone for functional focus, with additional access to natural language systems and qualitative reasoners, can further enhance explanation potential. See [7] for a discussion of these potentials.

The integration and use of the knowledge of deep models in compiled problem-solving systems has become increasingly prevalent. The intent of this research was to continue efforts in the development of a device- and domain-independent representation capable of modeling the core aspects of device understanding. The extended goal is a cognitive model of device understanding. Although this work was directed by the task of explanation, the representation is designed to provide the basic primitives and organization useful for a variety of problem-solving tasks. It is the hope that the research has made a positive step toward explicating some of the core aspects of device understanding in a way that provides contributions both toward a more complete model of device understanding and as an illustration of its potential uses for problem-solving systems.

Bibliography

- [1] B. Chandrasekaran and S. Mittal. On deep versus compiled approaches to diagnostic problem solving. *International Journal of Man Machine Studies*, 19:425-436, 1983.
- [2] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit. Diagnosis based on description of structure and function. In *Proc. National Conf. on AI*, pages 137-142. AAAI, 1982.
- [3] J. de Kleer. How circuits work. *Artificial Intelligence*, 24:205-280, 1984.
- [4] J. de Kleer and J. S. Brown. Assumptions and ambiguities in mechanistic mental models. In D. Gentner and A. S. Stevens, editors, *Mental Models*, chapter 8, pages 155-190. Lawrence Erlbaum Assoc., Hillsdale, New Jersey, 1983.
- [5] J. de Kleer and J. S. Brown. The origin, form and logic of qualitative physical laws. In *Proceedings of IJCAI-8*, pages 97-108. IJCAI, 1983.
- [6] A. Goel and B. Chandrasekaran. Understanding device feedback. Technical report, The Ohio State University, March 1988.
- [7] A. Keuneke. *Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions*. PhD thesis, The Ohio State University, Columbus, Ohio, 1989.
- [8] A. Keuneke and D. Allemang. Understanding devices: Representing dynamic states. Technical report, The Ohio State University, 1988.
- [9] B. Kuipers, de Kleer and Brown's "Mental Models": A critique. Technical Report 17, Tufts University, November 1981.
- [10] R. Patil. Design of a program for expert diagnosis of acid base and electrolyte disturbances. Thesis Proposal, 1979.
- [11] R. S. Patil. *Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis*. PhD thesis, MIT Lab for Computer Science, Cambridge, Massachusetts, 1981. TR-267.
- [12] C. Rieger and M. Grinberg. The declarative representation and procedural simulation of causality in physical mechanisms. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 250-255, 1977.

- [13] R. C. Schank and R. Abelson. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Assoc., Hillsdale, New Jersey, 1977.
- [14] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In J. L. Kolodner and C. K. Riesbeck, editors, *Experience, Memory, and Reasoning*, pages 47-73. Lawrence Erlbaum Assoc., Hillsdale, New Jersey, 1986.
- [15] C. A. Smith and A. B. Corripio. *Principles and Practice of Automatic Process Control*. John Wiley and Sons, Inc., New York, 1985.
- [16] G. Stephanopoulos. *Chemical Processing Control: An Introduction to Theory and Practice*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.
- [17] J. Sticklen, B. Chandrasekaran, and J. Josephson. Control issues in classificatory diagnosis. In *Proceedings of IJCAI-9*, pages 300-306. IJCAI, August 1985.
- [18] S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir. A model-based method for computer-aided medical decision-making. *Artificial Intelligence*, 11:145-172, 1978.
- [19] D. Weld. The use of aggregation in causal simulation. *Artificial Intelligence*, 30:1-34, 1986.

Appendix C

Explaining Control Strategies in Problem Solving

Explaining Control Strategies in Problem Solving*

B. Chandrasekaran

Michael C. Tanner

John R. Josephson

June 14, 1988

Laboratory for Artificial Intelligence Research
The Ohio State University

Abstract

Explaining the reasoning of knowledge-based systems involves issues of presentation, user modeling, dialog structure, and the system's self understanding. We concentrate on the last of these, how a system can understand its own problem-solving knowledge and strategy, since this provides the *content* of any explanation. Most current approaches to knowledge-based system construction require expressing knowledge and control at such a low level of abstraction that it is difficult to express explanations at a level appropriate to the problem. We propose the *generic task* methodology as a way of building knowledge-based systems which contain the basic explanation constructs at the appropriate level of abstraction. We give an example of explanation from a prototype system built using generic task methods.

1 Aspects of Explanation

In AI, as well as in many other fields, there is a growing body of research on the topic of explanation. This work falls mainly into two classes:

1. One major class of explanation involves explaining the world. There are two aspects to this:
 - (a) Several problem-solving systems have been built whose task can be abstractly characterized as explaining a set of data. Medical diagnosis, for example, can often be viewed as the generation of hypotheses to

*Research supported by Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010 under the Strategic Computing Program.

1 ASPECTS OF EXPLANATION

explain symptoms as is done by INTERNIST [1] and RED [2]. Also, DENDRAL hypothesizes molecular structures to explain spectroscopic data [3].

- (b) In both philosophy of science and cognitive science there is a concern about what kinds of descriptions are thought of as explanations of phenomena. Philosophers of science have proposed logical reconstructions of how theories can be considered explanations of observed phenomena [4,5]. From the cognitive viewpoint Schank has recently written about the issue of what kinds of explanation people deem acceptable in their everyday lives and developed a computational theory of how such explanations can be generated [6].
2. In another body of work the aim is to give programs the ability to *explain their decisions*. This is especially relevant to work on knowledge-based systems. The goal here is to use the system's knowledge to help a user understand how it reached its conclusions, help in debugging the knowledge base and problem-solving behavior, and perhaps to convince a user that the system's conclusions are reasonable.

These kinds of explanation are all relevant to AI. However, in this paper we are mainly interested in the second—explaining decisions.

The problem of explanation generation in knowledge-based systems can be broken down into three top-level functions:

Generating the Basic Content of Explanation Given a user query about some aspect of decision-making by the system, an information structure that contains the elements that make up an explanation needs to be generated. An essential element in the construction of such an explanation is how knowledge of the problem-solving task comes into play in explanation. The content of an explanation can be put together in two ways:

By "Introspecting" That is, by examining a record of its own problem solving activity and picking the appropriate traces that contain information for user query, or by retrieving portions of the knowledge base that were used in making the decision and thus can be used to support it. We need to know how a problem solver can *comprehend* its own problem solving activity.

By "Concocting" That is, by producing a justification which does not relate to how the decision was actually made, but independently makes the decision plausible. Such a *post facto* construction of a justification

1 ASPECTS OF EXPLANATION

or explanation is necessary when the problem solver has no access to a record of its own problem solving activity, or when the information contained in it is unnecessary or may be deemed incomprehensible to the user. The explanation may be a convincing argument that the answer is correct without actually referring to the process of deriving it, just as a mathematical proof persuades without representing the process the mathematician went through in deriving the theorem. Generating explanations of this sort is an interesting problem solving process in its own right as discussed by Wick, Thompson and Slagle [7].

Responsiveness: Shaping Explanations to Match User's Knowledge Not all the information in the explanation structure generated in the content process above will need to be communicated to the user. The user's goals, state of knowledge, and the structure of the dialog are used to *filter, shape, and organize* the output of the above process so that the explanation is responsive to the user's needs. This function often requires some form of *user modeling*.

Human-Computer Interface The two processes above produce all the information needed conceptually and logically for the needed explanation. However, presentation issues remain: how an appropriate human-computer interface displays and presents the information to a user in an effective way. What explanations are best presented in natural language, what in graphical form (such as "pie charts") are issues that would need to be faced for this function.

No matter how good the theories for responsiveness and interface functions, if the content of the explanation is inadequate or inappropriate, then correspondingly poor explanations will be presented well. Thus generating the correct content of the explanation is the problem of first priority in explanation generation. Accordingly, in our work, we have concentrated on one aspect of this problem, *viz.*, basing the content of explanation on introspection of the system's own problem solving behavior. This can be divided further into three types:

Type 1: Explaining why certain decisions were made or were not made. Portions of the data in a particular case are related to the knowledge for making specific decisions or choices.

Type 2: Explaining the elements of the knowledge base itself. For example, a system's compiled knowledge can be justified by linking it to deep knowledge from which it was derived. (For a discussion of issues related to "deep" and "compiled" knowledge see Chandrasekaran and Mittal[8].)

2 THREE GOOD IDEAS IN KNOWLEDGE SYSTEMS EXPLANATION

Type 3: Explaining the problem-solving strategy and the control behavior of the problem solver.

It should be noted that typically types 1 and 3 above involve the *run-time* behavior of a problem solver (and thus cannot in general be precompiled without running into combinatorial problems), while explanation structures for type 2 above can in principle be attached to the knowledge fragments at the time the knowledge base is put together.

These aspects of explanation are outlined in figure 1 with boxes around the areas which are the main interests in this paper. In our lab we have been working on the use of a functional representation for deep models of the domain [9] in producing type 2 explanations, relating the domain knowledge to problem-solving knowledge. We will only touch on this briefly here, see Chandrasekaran, Josephson, and Keuneke [10] for a more detailed discussion. The main focus of this paper we will be on how our theory of generic types of problem solving [11] is especially suited to building systems which can explain their control strategy (type 3 above). Systems built using this *generic task* approach are composed of knowledge-level agents with well-defined problem-solving roles. Thus, they are also capable of producing type 1 explanations, how data match knowledge, based on the memory of each agent for its own problem-solving history. We have implemented a knowledge-based planning assistant to test these ideas on explanation.

2 Three Good Ideas in Knowledge Systems Explanation

The history of explanation in knowledge-based systems can be summarized by three good ideas corresponding to the types of explanation given in the previous section. The first was due to Davis, Shortliffe, and the others who worked on the original MYCIN system [12]. The essential idea is that a trace of problem-solving activity at the level of the implementation—for MYCIN this would be the rule architecture—can be used to give explanations about what a system did. MYCIN's explanation facility examined its behavior at the level of its rule architecture. This facility could answer questions from the user about how and why certain conclusions were reached. MYCIN's explanations were entirely expressed in terms of rules and goals. The question "WHY?" was interpreted as, "Which rule needs this datum, and what is the consequent in the rule?" Thus, MYCIN produced type 1 explanations by giving a rule trace. For certain purposes, such as debugging, this is entirely appropriate.

With his XPLAIN system [13], Swartout introduced the second good idea. A knowledge-based system has task-specific goals and problem-solving knowledge

2 THREE GOOD IDEAS IN KNOWLEDGE SYSTEMS EXPLANATION

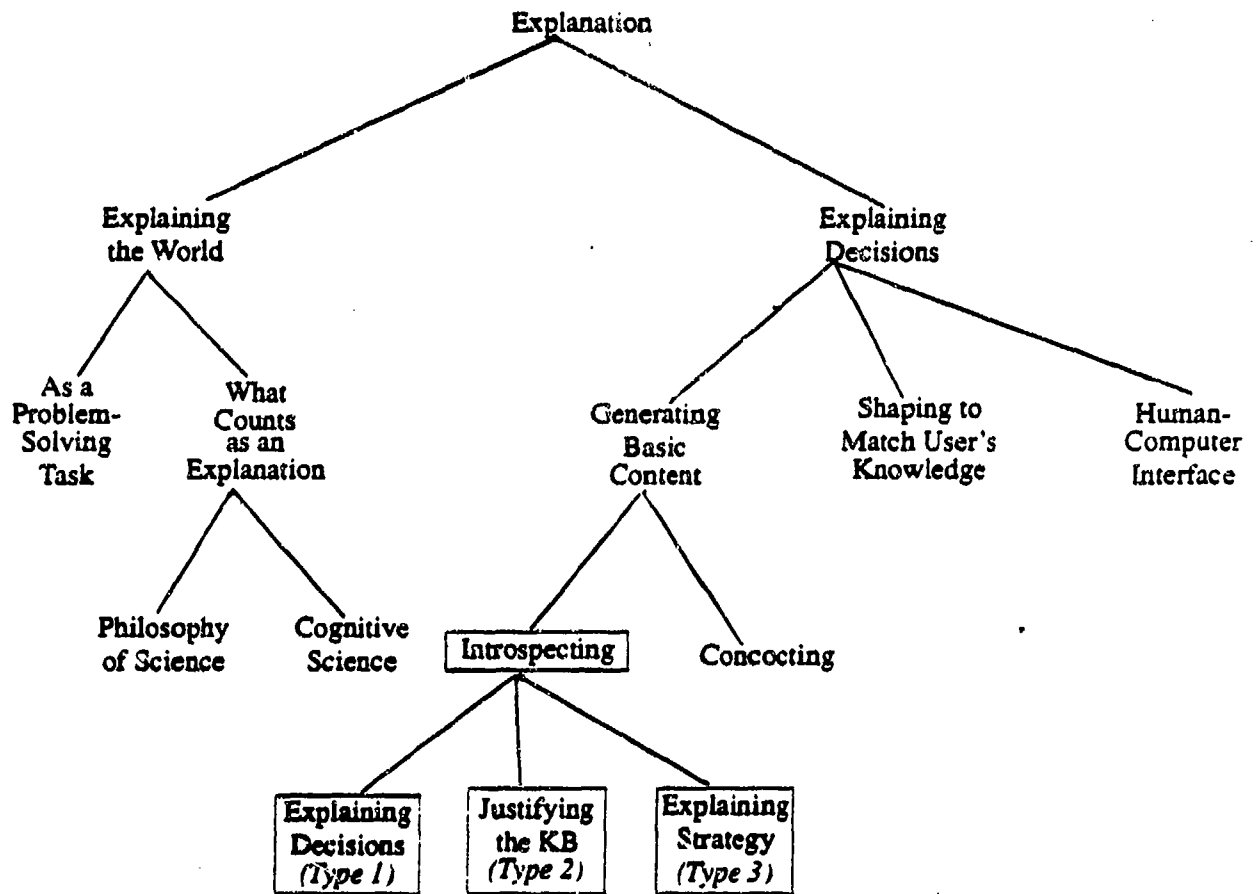


Figure 1: Aspects of Explanation

2 THREE GOOD IDEAS IN KNOWLEDGE SYSTEMS EXPLANATION

which can be thought of as *compiled* from more general domain knowledge. If a trace of the compilation is remembered then rules in the system can be justified in terms of the deeper knowledge. XPLAIN is able to use deep knowledge, called the "domain model," and a representation of problem-solving control strategies, called "domain principles," to compile a knowledge-based system. Thus, the control strategy can be examined for analyzing the system behavior and the deep model can be used to justify the system's rules. In terms of the explanation types given in section 1, this is type 2 explanation.

Clancey noted that a knowledge-based system typically performs a task at a higher level of abstraction than the goal-subgoal level of a rule base [14]. But MYCIN had explicit representation only of the rules, and not of the problem-solving *strategies* that may be implicitly encoded in the rule formalism by the system designer. Thus, it could not answer "WHY?" questions that needed to be interpreted strategically. However, as Clancey points out, if a system's behavior is *represented* at the task level, it can produce *explanations* at the task level. This is the third important idea on explanation. NEOMYCIN solves the same diagnosis problem as MYCIN but represents the diagnostic task explicitly. It contains such diagnostic operators as "establish hypothesis space" and "explore and refine" which represent the diagnostic strategy and in terms of which it can explain its problem-solving activity. Thus, NEOMYCIN can give strategic explanations which describe its higher level goals, i.e., type 3 explanation.

Virtually all knowledge-based systems which produce explanations use one or more of these ideas.¹ Many problem solvers have been built using EMYCIN [15] so their explanation is necessarily similar to MYCIN's trace of run-time behavior. But many systems which are not built with EMYCIN, or not even with rules, still explain using a trace. For example, ABEL [16] reasons using a causal network but it explains by translating pieces of the net into English, i.e., tracing its actual reasoning. ABEL also contains several levels of detail in its causal net and so it can justify its reasoning at one level by reference to deeper levels, thus providing type 2 explanation. Another approach to justifying knowledge is found in the Integrated Diagnostic Model [17]. It contains deep and surface models of a domain and is capable of justifying the surface knowledge by reference to the deep knowledge. Unlike XPLAIN, in IDM the connections between the levels are made by the system builder. For strategic explanations consider SOPHIE [18]. One of the important goals of SOPHIE was to explain problem-solving strategies. However, lacking a task-level model of problem solving as found in generic task theory [11] and in NEOMYCIN, this is accomplished by hand-encoding the strategic explana-

¹The main exception is numerical based systems, e.g., Bayesian decision systems, in which explanation comes from a sensitivity analysis of the formulae which produce the answer.

3 A FRAMEWORK FOR EXPLANATION

MYCIN: Explanation of decisions at the level of system knowledge.

NEOMYCIN: Explanation of problem-solving strategy at the task level.

XPLAIN: Justification of knowledge by reference to a deeper level.

Figure 2: The Origin of Three Important Ideas

tions.

Our own work fits directly into the spirit of these three ideas, summarized in figure 2, though we differ in some ways. We want our systems to explain themselves at the architecture level in a way that is appropriate to the problem-solving task that they are performing and we think it is necessary to have a deep model of the domain which can be accessed by the system for justifying its knowledge. In this paper we set up a framework for explanation which we hope will clarify these points of similarity and difference and provide a common ground for discussion of the topic of explanation. In addition, we propose an approach to building knowledge-based systems in which explanations can be naturally derived.

3 A Framework for Explanation

3.1 Types of Explanations

In section 1 we said that explanation of problem-solving activity can be categorized into *three distinct types*, viz., Type 1, how the data matches the local goals; Type 2, how the knowledge itself can be justified; and Type 3, how the control strategy can be justified. These three types correspond to the structures that need to be examined to construct the explanation. In this section we elaborate the description of these types and give some examples.²

Type 1 Explanation relates the actual problem-solving behavior to a problem state or data describing the problem. This involves examining appropriate fragments of the run-time behavior of the system. For example, in a medical diagnosis system,

User: Why do you say the patient has cholestasis?

²The examples in this section are created for expository purposes only and are not meant to represent actual explanations.

3 A FRAMEWORK FOR EXPLANATION

System: Because the patient has high blood bilirubin, jaundice, and X-rays suggest an obstruction in the biliary duct.

Or, in an economic planning consultant.

User: Why do you conclude that a tax cut is appropriate here?

System: Because its preconditions are high inflation and trade deficits, and current conditions include these factors.

These explanations are of how, for this *particular* instance of the problem, problem specific data matched pieces of the knowledge base and certain conclusions were drawn. It is important to note that, e.g., it may be possible to conclude cholestasis from a number of different possible combinations of data. The user wants to know which data combination was present in this particular problem. This requires keeping a trace of the problem-solving behavior, examining it, and constructing the explanation from the trace.

The user may not be satisfied with this level of explanation. He may ask.

"Why does high blood bilirubin indicate cholestasis? Must it occur in conjunction with jaundice?"

Or,

"Why is a tax cut a good idea for shrinking trade deficits?"

The answer to this does not involve the particular situation at hand. The system is being asked to explain portions of the knowledge base itself. We call this *Type 2 Explanation*.

Type 2 explanations are concerned with explaining elements of the knowledge base. These explanations will often be based on how the knowledge was derived. There are at least four ways of obtaining knowledge for problem solving, each with its corresponding types of explanation.

1. By directly being told: The knowledge can only be justified by appeal to authority, e.g., "Text book, p. 85."
2. By generalizing from examples: "68% of the time when a tax cut was tried, the trade deficit went down," "the last time a patient had these symptoms it turned out to be AIDS," etc.
3. By explanatory inference: For example, suppose a system contains the rule that, given certain symptoms, a certain infectious organism should be hypothesized. This can be justified by the knowledge that if the disease were

3 A FRAMEWORK FOR EXPLANATION

present it would explain the symptoms. Further justification might include a discussion of the medical science history of recognizing the group of symptoms as a distinct disease and identifying the infecting organism. Note that the explanatory inference (that this organism is the cause of, hence explains, the symptoms) may be compiled out of the problem-solving knowledge; that is, the system may be unable to actually make the explanatory inference and still be able to solve problems. However, it will need access to the inferential history in order to justify the knowledge.

4. By derivation from a deeper understanding of the domain: "Tax cuts generally encourage savings, stimulate investment, and increase production, which decreases prices, makes goods made here attractive, increases exports, and thus reduces trade deficits." Notice that this form of reasoning is not *needed* for concluding a tax cut is appropriate if the knowledge base contains the pieces of knowledge that relate tax cuts to shrinking trade deficits. It is nevertheless useful to keep such knowledge available so that it can be appropriately used for explanation.

Some of our work, reported briefly in this paper and in more detail elsewhere [10], has been concentrated in the fourth category above, viz., reasoning from deep models. In particular, we have developed an approach in which an agent's understanding of how a device works can be used to derive fragments of diagnostic knowledge. Thus a diagnostic knowledge-based system can justify a decision by tracing how a piece of diagnostic knowledge was derived from an understanding of the structure and function of the device involved.

Consider the following interactions with a medical system.

User: Why didn't you consider portal hypertension in this case?

System: Because I had ruled out circulatory diseases, portal hypertension is a special case of circulatory diseases, and *my strategy is not to consider special cases when I have ruled out the general case.*

Or an economic planning consultant.

User: Why aren't you suggesting increased tariffs as a way of decreasing trade deficits?

System: Because that plan involves political costs. *My strategy is to consider politically easier plans first.*

3 A FRAMEWORK FOR EXPLANATION

In these explanations part of what is being explained is the control strategy of the knowledge-based system, *Type 3 Explanation*.

This form of explanation can account for "why not?" as well as "why" questions of certain types. What is needed is an ability to *abstract* the control strategy, and *match* portions of it to the situation. The actual explanation typically involves a combination of explanations of Type 1 and Type 3. E.g., the italicized portions of the explanations above are of Type 3, and the rest are of Type 1.

Note that in both Swartout's XPLAIN and in Clancey's work on explanations in rule-based systems, a goal-subgoal tree is an essential element in the generation of strategic explanations. Our theory of types of problem solving will be shown in Section 3.2 to enrich the explanatory vocabulary by identifying the types of goals and subgoals.

We have described three types of explanation:

Type 1: Trace of run-time, data-dependent, problem-solving behavior.

Type 2: Justifying a piece of knowledge by deriving it from a domain model.

Type 3: Understanding the control strategy used by the program in a particular situation.

In figure 3 we outline this framework of explanation. Our theory of problem-solving types (which is also a theory of control structure types) is applicable to Type 3. In the next two sections we outline this theory and show how it helps in explanation.

3.2 Types of Problem Solving and Explanation

In this paper we can only briefly outline our theory of problem-solving types, called generic tasks (these ideas are presented in greater detail by Chandrasekaran [11]). A central idea of generic tasks is that the function of the task (i.e., the input-output behavior), the kind of knowledge needed to perform it and the kinds of inferences that are appropriate for it are all specified together. To understand how this contrasts with the traditional rule-based paradigm, consider as an example the R1 system [19]. R1 itself can be thought of as performing a limited type of design problem solving which can be implemented as a linear sequence of subtasks. That is, at the level of the task, the required knowledge is one of how to decompose the design task into subtasks, and the underlying control strategy is one of linear execution of subtasks. R1 is implemented using the rule language OPS5, and uses a "forward-chaining" inference mechanism. The explicitness of the above description of the knowledge and inference of R1's task is missing at the level of the OPS5 code which

3 A FRAMEWORK FOR EXPLANATION

Type of Explanation	Source	Comments
Problem-Solving Behavior	Problem Solver (i.e., Expert System)	
Problem-Solving Strategy, Control Decisions	Generic Task	Problem solvers can explain their strategy if they contain the proper connections to their generic task
Justification of Problem-Solving Knowledge	Functional Representation (or some source of deep knowledge)	Problem solvers can justify their problem solving knowledge if they are properly connected to the relevant deep models of the domain.

Figure 3: A Framework for Explanation

implements R1. Similarly, at the level of EMYCIN, which is the implementation level language for MYCIN, one sees rules and backward-chaining, whereas the real task MYCIN performs is best understood as "heuristic classification" [20]. The generic task methodology is based on reducing the gap between the implementation level and the intrinsic task level. The advantages of this approach for system design and knowledge-acquisition are described in Chandrasekaran [11], and Bylander and Chandrasekaran [21]. The main topic of this paper is the leverage that this methodology gives for explanation of control strategies.

In our work on knowledge-based reasoning in medical and mechanical systems, we have identified several such generic tasks. Here we briefly summarize two of them.

Hierarchical Classification. Classify a description of a situation as one or more elements in a *classification hierarchy*. The classificatory knowledge is *distributed* among concepts in this hierarchy. Each conceptual *specialist* contains knowledge that helps it determine whether the concept it stands for can be *established* or *rejected*. Problem solving is top down. Each specialist, when called, tries to match its description against the situation description. If it succeeds, it calls its successors, which repeat the process. Otherwise, it rejects and all its successors are also automatically rejected. This control strategy is called *Establish-Refine*, and results

3 A FRAMEWORK FOR EXPLANATION

in a specific classification. (This account is a simplified one. The reader is referred to Bylander and Mittal[22] for details and elaborations.)

Design by Plan Selection and Refinement. Design an object satisfying certain specifications. *Specialists* corresponding to components are organized in a hierarchy mirroring the object structure. Each specialist has *plans* which can be used to make commitments for some parameters of the component. Control is top down in general. The following is done recursively until a complete design is worked out: A specialist is called, the specialist chooses a plan, based on some specification, which instantiates some parts of the design and calls further specialists to fill in other parts of the design. Failures are passed up the specialist hierarchy until appropriate changes are made by higher level specialists so specialists that failed may succeed on a second try. This problem-solving type is discussed in more detail in Section 4.1.

State Abstraction. Given a change in some state of a system, provide an account of the changes that can be expected in the functions of the system. Knowledge is distributed in conceptual specialists corresponding to systems and subsystems. These specialists are connected in a way that mirrors the way the system is put together. Control is basically bottom-up, following the architecture of the system/subsystem relationship. The changes in states are followed through, the functions at the desired level of abstraction are obtained. This is useful for reasoning about consequences of actions on complex systems.

Knowledge-Directed Information Passing. Given attributes of some datum, it is desired to obtain attributes of some other conceptually related datum. The data is organized as a *frame hierarchy* where each frame is a specialist in knowledge-directed information passing for a concept. A specialist, when asked for the value of an attribute, first checks to see if the actual value is known, if not it then uses inheritance relationships to determine whether the value can be obtained by inference from the values of appropriate attributes of its parent or children, and if that fails it then uses any demons that may be attached to the slot to query other concepts in other parts of the hierarchy for values of their attributes. If none of this succeeds and if it is appropriate, a default value is produced. This is useful for inferring a medical datum which is unavailable but needed in diagnosis, from another which is available [23].

3.3 Role of Problem-Solving Types in Type 3 Explanation

A typology of problem solving, such as the one outlined in the previous section, with its associated goal types, is very useful in explaining the control strategy of the problem solver. Let us take some examples.

3 A FRAMEWORK FOR EXPLANATION

Hierarchical Classification

User: Why did you wish to know if the patient had been exposed to anesthetics?

System: Because I was trying to *establish* hepatitis.

User: Why?

System: Because I had already established that the patient had liver disease. Now I am trying to subclassify the case, and hepatitis is one of the subcategories. If I can *reject* hepatitis, I can *reject* all its subclasses.

Design by Plan Selection and Refinement

User: Why did you choose the standard table top plan?

System: Because I am trying to complete the specification for the table. A part of *refining* the table specification requires designing the table top. The table top specialist selected the standard table top plan because all the requirements were normal.

User: What will you do if the standard table top plan fails?

System: The table top specialist will *select* the alternate table top plan.

User: What if it fails?

System: The table specialist will *redesign* the table by weakening the constraints on the table top.

State Abstraction

User: Why do you wish to know the engine speed?

System: Because I am trying to decide the *change in function* of the cooling system when the thermostatic valve is stuck closed. Engine speed is a datum I need for this.

User: Why are you considering the cooling system.

System: Because the thermostatic valve is a *component* of the cooling system.

3 A FRAMEWORK FOR EXPLANATION

Knowledge-Directed Information Passing

User: Was the patient exposed to anesthetics?

System: Did he have any surgery?

User: Why do you ask?

System: I do not have a direct record of anesthetic exposure. I could not *inherit* a value for this attribute since there is no direct record of any drug being given. I was also unable to infer no exposure by *ruling out all types* of anesthetic. But if the patient had surgery, then anesthetic can be reasonably inferred.

In these examples, the italicized terms represent the problem-solving goal that is being pursued. For example, *establish* and *reject* are a few of the goals of the *Establish-Refine* control strategy associated with the problem-solving type *hierarchical classification* (see section 3.2). We believe that terms such as these enrich one's understanding of the behavior of the system. In addition, these examples, while based on the appropriate fragments of control knowledge, nevertheless combined the control terms with the state of the problem solving. Thus the actual generation of the explanation requires a mixture of Types 1 and 3 in the list of explanatory types.

So far in our research we have identified six generic types of problem-solving behavior underlying knowledge-based problem solving. Hierarchical classification, a kind of design, state abstraction, and knowledge-directed information passing, described in section 3.2, are examples. Each type of problem solving uses knowledge in a certain form and has an associated family of control strategies. These two features, kinds of knowledge and control strategies, constitute an architecture for a particular problem-solving type. A high level programming language can be designed corresponding to each architecture. A set of such languages would be used in building knowledge-based problem-solving programs. Note the similarity of this idea to the way the programming language EMYCIN is related to the rule architecture of MYCIN. Elsewhere [11] we have described in such a family of high level languages for constructing knowledge-based systems.

When a system is built using a generic task programming language, the trace of its problem-solving behavior at the architecture level is automatically couched in terms of the strategic goals of its control strategy (e.g., *establish*, *reject*, *select*, and *redesign* in the above examples). The control strategy itself is explicitly represented in the language so it will naturally be included in a trace. Thus, the generic task approach allows us to achieve explanation of problem-solving strategy by using a trace of the problem solver. In the next section we describe a system built using a generic task language and give examples of its explanations.

4 IMPLEMENTATION EXAMPLE

4 Implementation Example

4.1 Mission Planning and Class 3 Design

We have chosen the "routine planning" task for Offensive Counter Air (OCA) missions dealt with by the KNOBS system [24] as a task for which to build a prototype. Our prototype is called the Mission Planning Assistant (MPA). We emphasize that MPA is incomplete as a mission planner but it has served its purpose as a vehicle for exploring these ideas.

KNOBS plans by template instantiation—a process of filling in the slots of a frame with acceptable values. The order in which the slots are considered is defined in advance by the plan template, and is determined by the expert's domain planning knowledge. Slot values are accepted or rejected based upon constraint satisfaction. KNOBS associates a generator with each slot to enumerate potential values.

A problem with KNOBS, recognized by its designers, was that it did not explicitly represent much problem-solving expertise for planning. Thus the KNOBS mechanism does not allow for explaining the planning control strategy. We approach this problem from the generic task viewpoint which, by tailoring the control strategy to the problem-solving task, provides the structures necessary for explaining strategies.

We treat the OCA mission as an abstract device to be designed. The planning of the mission involves a process similar to the process a designer undergoes when faced with a complex device to design. An overview of the design process will illuminate this analogy. For a more comprehensive description see Brown and Chandrasekaran[25].

Design as a problem-solving activity potentially involves creativity, many different problem-solving techniques, and many kinds of knowledge. Goals are often poorly specified, and may change during the course of the activity. However, a spectrum of design classes can be identified, varying from completely open-ended activity to the most routine, depending on what sort of knowledge is available. We identify one of the most routine forms of design as "Class 3 Design." In this class of activity complete knowledge is assumed to be available both of the components which need to be designed, and of potentially useful design plans for each component. This does not imply that the design process itself is simple, nor that the components so designed must be simple. It appears that a significant portion of everyday activity of practicing designers falls into this class. Class 3 design is a hierarchical planning task since devices typically decompose into components and subcomponents, and thus the design tasks decompose. Each level in the hierarchy makes some design commitments, and the design is further refined in the lower

4 IMPLEMENTATION EXAMPLE

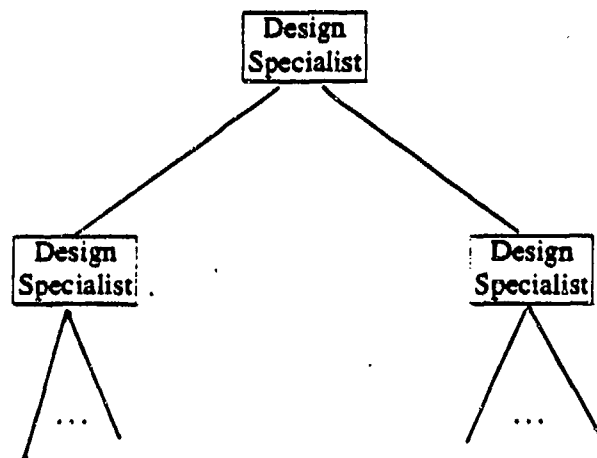


Figure 4: Organization of a DSPL Problem Solver

levels. In order to explore this class of design problems, the Design Structures and Plans Language (DSPL) was developed [25].

A design problem solver in DSPL (see figures 4 and 5) consists of a hierarchy of *specialists*, each responsible for a particular portion of the design. Specialists higher up in the hierarchy deal with the more general aspects of the device being designed, while specialists lower in the hierarchy design more specific sub-portions of the device. The organization of the specialists, and the specific content of each, is intended to capture the designer's expertise in the problem domain.

Each specialist in the design hierarchy contains the design knowledge necessary to accomplish a portion of the design. There are several types of knowledge represented in each specialist, but for simplicity only three are described here. First, explicit *design plans* in each specialist encode sequences of possible actions to successfully complete the specialist's task. Second, specialists have *design plan sponsors* associated with each plan to determine the appropriateness of the plan in the run-time context. And third, each specialist has a *design plan selector* to examine the run-time judgments of the sponsors and determine which of the specialist's plans is most appropriate to the current problem context.

Control in a DSPL system proceeds from the top-most specialist in the design

4 IMPLEMENTATION EXAMPLE

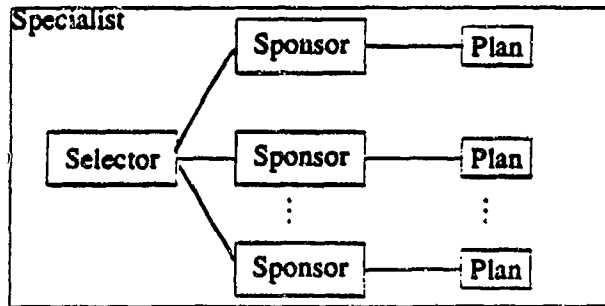


Figure 5: Inside a DSPL Specialist

hierarchy to the lowest. Each specialist selects a plan appropriate to the requirements of the problem. This plan is executed by performing the design actions it specifies. This may include computing and assigning specific values to attributes of the device, checking constraints, or invoking sub-specialists to complete another portion of the design. (We have called this process "design by plan selection and refinement.")

Mission planning is a class 3 design task. The problem can be decomposed into the design of subcomponents of the mission plan. In the domain of mechanical device design, the design of a device is decomposed into the design of sub-assemblies and their components, etc., where each sub-assembly or component can be designed in a fairly independent fashion. In the mission planning domain, the OCA is decomposed into various parts of missions where each part can be planned relatively independently of the others. Using DSPL as a mechanism for representing the necessary knowledge, the MPA system closely mirrors these ideas.

The DSPL control structure provides the framework for a comprehensive explanation facility. In addition to the necessary ability to examine particular attributes of a mission plan, the control structure provides the ability to examine the problem-solving strategies of the system.

4.2 Explanation in the Mission Planning Assistant

Our implementation of explanation in MPA is based on the organizing principle that *the agent which makes a decision is responsible for justifying it*. MPA is built in DSPL so the agents which contribute to the final plan include: specialists, plans,

4 IMPLEMENTATION EXAMPLE

selectors, and sponsors. In the present implementation there are some 200 of these agents, though not all of them contribute to any particular plan. All of these agents have well-defined roles so explanation of any one agent's problem-solving decisions can be given in terms of the *goals of the agent which uses it*, and the *function of the agents it uses*.

The final answer produced by the MPA is a list of attribute-value pairs as in KNOBS. That is, a list of the form:

```
Target = Berlin
Aircraft Type = F-111
Number of Aircraft = 6
...
```

We have decided to concentrate on questions of the form, "How was it decided?" which can be asked of the value of any attribute. For example, selecting F-111 in the above list would initiate a dialog on the question of how MPA decided to use F-111 as the value of **Aircraft Type**. Questions of this form directly ask about the problem solving which led to final decisions. While the answers do not exhaust the kinds of things a problem solver can say about its actions, we believe that this represents an important basic type of explanation, important for debugging the knowledge base as well as for knowing whether the system's recommendations can be trusted.

As an example we consider a single agent type, namely, plan sponsors, and the explanations they can produce. Explanations from agents of the other types are produced similarly.

Sponsors operate in the context of a DSPL specialist. A specialist's job is to design a component of the device, in order to do this it has available several alternative courses of action, called plans. Each plan is appropriate under some conditions and not others. So associated with each plan is a sponsor which matches characteristics of the plan to information about the problem at hand, and produces a measure of how useful the plan will be on a scale of: Ruled-Out, Unsuitable, Suitable, and Perfect. The specialist then chooses a plan based on what the sponsors return. The code for a sponsor in MPA is given in figure 6. This is a sponsor for a plan which uses A-10s (an aircraft type) on the mission. It first sets some local variables by looking them up in the data base (using *KB-FETCH*, a data base access function which simply returns the value of an attribute). The *TABLE* construct is a group of rules which all depend on predicates of the same values. For example, the table setting the variable *conditions* contains three rules which depend on the values returned by the functions *night* and *weather*. The first rule requires *night* to return *F*

4 IMPLEMENTATION EXAMPLE

```

( SPONSOR A-10
  (SETQ target (KB-FETCH TARGET))
  (SETQ timeOverTarget (KB-FETCH TIMEOVERTARGET))
  (SETQ threat
    (TABLE (airborne) (AAA) (SAM)
      (IF T ? ? THEN UNSUITABLE)
      (IF ? T ? THEN UNSUITABLE)
      (IF ? ? T THEN UNSUITABLE)
      (IF ? ? ? THEN PERFECT)))
  (SETQ conditions
    (TABLE (night) (weather)
      (IF F FULL THEN UNSUITABLE)
      (IF F PARTIAL THEN SUITABLE)
      (IF F GOOD THEN PERFECT)))
  REPLY
    (TABLE conditions threat
      (IF UNSUITABLE ? THEN RULE-OUT)
      (IF ? UNSUITABLE THEN RULE-OUT)
      (IF SUITABLE ? THEN SUITABLE)
      (IF ? ? THEN PERFECT)))

```

Figure 6: DSPL code for a Design Plan Sponsor

and *weather* to return *FULL*.³ If that is the case, *conditions* will be *UNSUITABLE*. The symbol '?' in the tables represents a predicate which is always true. The table is finished when one rule matches. The code following *REPLY* is the main function of the sponsor, i.e., rating the suitability of a plan.

An explanation produced for this sponsor is given in figure 7. This explanation could be produced because sponsors are problem-solving agents of a known type that follow a known form in their implementation. Values for the local variables are given, those fetched from the data base are not justified while those determined by tables are given justification. The final *REPLY* is used to determine the actual decision made by the sponsor. This explanation describes how the data matched problem-solving knowledge at run-time (type 1 explanation). MPA does not have an explicit representation of its control strategy; however, explanation of strategy

³The reader might think *FULL* and *PARTIAL* are strange values for *weather* but these terms come from the domain. Here *FULL* means "full cloud cover", *PARTIAL* means "partial cloud cover."

4 IMPLEMENTATION EXAMPLE

The context of *selecting an aircraft to consider for the mission* determined that:

- target is **BrandenburgSAM**
- timeOverTarget is 1300
- threat is **UNSUITABLE** because:
 - SAM is **TRUE**
- conditions are **PERFECT** because:
 - weather is **GOOD**

The value of plan A-10 is **RULE-OUT** because:

- threat is **UNSUITABLE**.

Figure 7: Explanation for a Design Plan Sponsor

is possible. The italicized phrase at the beginning of the explanation comes from the agent, a specialist, which uses the sponsor to do its job. The user of MPA can follow up by getting an explanation from that specialist which would spell out the sponsor's context in more detail by giving the problem-solving strategy MPA was pursuing at the point where the sponsor was invoked. Thus, the explanation for the calling agent gives a problem-specific explanation of the control strategy in service of which the agent was called (type 3 explanation).

4.3 Understanding a Mission Plan

In section 3.1 we describe type 2 explanation, explaining elements of the knowledge base. In the mission planning domain, consider the question, "Why was an F-111 used?" In the previous section this question would be interpreted as, "Why was an F-111 used *instead of* any other aircraft?" An alternate interpretation could be, "Why was an F-111 *used* in the mission plan?", i.e., inquiring about the role played by the F-111 in the mission plan. A good response here might be, "The F-111 is an aircraft. Aircraft are used in OCA's because they can fly and deliver the ordnance. These functions are used to get to the target and destroy it." In other words, the question asks for, and the response gives, an explanation of a part of the plan knowledge base, i.e., type 2 explanation. This type of explanation requires a different understanding of the domain than the planner has available in its plan-

4 IMPLEMENTATION EXAMPLE

ning knowledge. What is needed is a representation of *how the plan works*. For this we use the *functional representation of devices* as given by Sembugamoorthy and Chandrasekaran[9]. A device is any structure (concrete or abstract) which serves a purpose. Thus, a plan can be viewed as an abstract device in that it has components which act together in order to achieve a desired goal.

A part of the *functional representation* of the abstract device OCAMission which describes the main function of an OCA—to destroy a target—is shown below.

```
FUNCTION: DestroyTarget:
    TOMAKE Destroyed(Target)
    IF Functional(Target)
    PROVIDED Operational(Flight)
    BY OCAPlan
```

This description indicates that the plan, OCAMission, has a *function* called DestroyTarget. This function is appropriate *if* a target is operational (functional). When this function is used, the target will be destroyed *by* a behavior called OCAPlan. This behavior should succeed in accomplishing the goal of target destruction *provided* the flight is operational throughout the behavior.

The *behaviors* of a device describe the manner in which its functions are accomplished by using the functions of components, generic knowledge, and sub-behaviors. For example, the behavior for OCAPlan is described by a chain of events as given in figure 8. This structure is meant to represent the temporal sequence (from top to bottom) of states which occur as a result of actions taken. The diagram thus indicates that the OCAPlan's behavior begins when a Target is in a Functional state. Then the function PrepareFlight of the component AirBase makes the Flight Prepared. Upon achieving this state, the function OffensiveAir of the component Flight is used to Destroy the Target. And so on.

The functional representation can be used to give type 2 explanations. Here we give an example of the kind of answers which can theoretically be given using this representation. Explanation responses are built from the functional primitives. In the context of the top level device, OCAMission, the following interaction might occur,

User: What does the function DestroyTarget do?

System: The function DestroyTarget ensures that the target is destroyed. It can be used if the target is functional and provided OCAPlan satisfies the constraint that the Flight remain Operational.

4 IMPLEMENTATION EXAMPLE

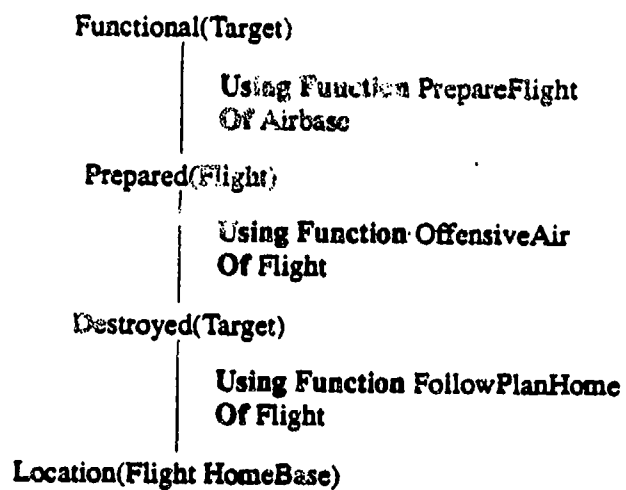


Figure 8: A Behavior in MPA

5 RECAPITULATION AND CONCLUSION

(See the description of DestroyTarget earlier in this section.) For more details and actual examples the reader is referred to Chandrasekaran, Josephson, and Keun-cke[10].

5 Recapitulation and Conclusion

In section 2 we characterized previous work on explanation as producing three good ideas. These are:

1. Meaningful explanation can be given at the architecture level.
2. Justification of knowledge requires access to deep models.
3. Some kinds of explanation require explicit representations of problem-solving strategies.

In section 3.1 we described three types of explanation:

Type 1: Explaining how data matches the knowledge.

Type 2: Explaining the knowledge itself.

Type 3: Explaining the problem-solving strategy.

In order to generate explanation of types 1 and 3 *at the appropriate level of abstraction*, the problem-solving process needs to be represented at what we have called the *generic task level*. Most current approaches to knowledge-based system construction use knowledge-representation languages and control primitives at too low a level of abstraction (the rule/frame/logical formula level). This makes both system design and explanation difficult, since the system designer has to transform the problem into a low level implementation language and explanation requires translating back to the problem level. We have identified a set of higher level building blocks in terms of which systems can be conceptualized, designed, and implemented. The basic explanation constructs are then available closer to the conceptual level of the user than they would be if they had to be extracted from the implementation language level. This point of view has led us to propose a new approach to the design of knowledge-based systems, namely, *generic tasks*. With each generic task we have developed a high level programming language which contains the task's problem-solving control strategy (e.g., DSPL [25] and CSRL [22]). When a system built using these languages gives a trace (type 1 explanation) it also describes its strategy (type 3 explanation) because it uses terms which denote task-specific goals and methods.

5 RECAPITULATION AND CONCLUSION

The MPA prototype system has demonstrated the viability of the generic task approach to explanation in knowledge-based systems. However, it is limited both in the scope of its problem-solving knowledge and in its ability to generate complete explanations. In spite of its limitations we believe MPA has fulfilled our original goal of showing how explanation can naturally be derived from deep models and an understanding of the problem-solving task.

The compilation approach proposed by Swartout is orthogonal to the approach we give here and very much compatible with it. The generic task idea can be thought of as providing a vocabulary of problem-solving goals. When these goals are embodied in programming languages for building knowledge-based systems, they are easily available for use in explanation. A compiler such as the one in XPLAIN could have the generic tasks built into it so that the resulting system would have the explanatory capabilities we describe.

In comparison with Clancey's work there are interesting points of contact and difference. We both emphasize the importance of a higher level vocabulary of goals for proper generation of explanation. Our work provides part of this vocabulary in a manner that can be naturally integrated with an approach to building knowledge-based systems using higher level building blocks.

In this paper we have not discussed the kind of explanatory problem that still remains even after the explanation at the more abstract task-level such as the one provided by the generic task approach. There is still a gap between the latter level and the level of the actual problem solving task. For example, consider a diagnosis system developed in our lab called RED [26]. This system is built out of four of the generic tasks (see section 3.2)—concept matching, classification, knowledge-directed information passing, and abductive assembly—which jointly perform a task similar to heuristic classification [20], though at a higher level it is actually doing diagnosis. The ideas set forth in this paper could be applied to RED and it would then be capable of generating explanations at the generic task level. While this is a significant improvement over the explanations given by MYCIN it is still not at the level of diagnosis. There is a need to explain how the system's actions and decisions meet the goals of diagnosis. We have recently been working on explanations of this sort, explaining how the system's conclusion relates to the goals of the problem-solving task [27]. This will result in type 3 explanation of a different sort than is described in this paper.

Acknowledgments

We gratefully acknowledge the assistance of Mitre Corporation for providing the KNOBS source code and taking the time to help us understand KNOBS. People

REFERENCES

who contributed to the MPA project include Dean Allemang, Matt DeJongh, Ron Hartung, and Dave Herman. In addition, the functional representation of the mission plan, on which section 4.3 is based, was done by Anne Keuneke.

References

- [1] H. E. Pople, "The formation of composite hypotheses in diagnostic problem solving," in *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, (Cambridge, MA), pp. 1030-1037, August 22-23 1977.
- [2] J. R. Josephson, B. Chandrasekaran, J. R. Smith, and M. C. Tanner, "A mechanism for forming composite explanatory hypotheses," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-17, pp. 445-454, May/June 1987.
- [3] B. G. Buchanan and E. A. Feigenbaum, "Dendral and Meta-Dendral: Their applications dimension," in *Readings in Artificial Intelligence*, (B. L. Webber and N. J. Nilsson, eds.), pp. 313-322, Palo Alto, CA: Tioga, 1981.
- [4] Aristotle, "Posterior analytics," in *The Basic Works of Aristotle*, (R. McKeon, ed.), pp. 108-186, New York, NY: Random House, 1941.
- [5] C. G. Hempel, *Aspects of Scientific Explanation*. New York, NY: Free Press, 1965.
- [6] R. Schank, *Explanation Patterns*. Hillsdale, NJ: Erlbaum, 1986.
- [7] M. R. Wick, W. B. Thompson, and J. R. Slagle, "Knowledge-based explanation," TR 88-24, Computer Science Dept., Univ. of Minnesota, Minneapolis, MN, March 1988.
- [8] B. Chandrasekaran and S. Mittal, "On deep versus compiled approaches to diagnostic problem-solving," *International Journal of Man-Machine Studies*, vol. 19, pp. 425-436, November 1983.
- [9] V. Sengamoorthy and B. Chandrasekaran, "Functional representation of devices and compilation of diagnostic problem solving systems," in *Experience, Memory and Reasoning*, (J. L. Kolodner and C. K. Riesbeck, eds.), pp. 47-73, Hillsdale, NJ: Erlbaum, 1986.
- [10] B. Chandrasekaran, J. Josephson, and A. Keuneke, "Functional representations as a basis for generating explanations," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, (Atlanta, GA), pp. 726-731, October 1986.

REFERENCES

- [11] B. Chandrasekaran, "Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design," *IEEE Expert*, vol. 1, pp. 23-30, Fall 1986.
- [12] R. Davis, "Teiresias: Applications of meta-level knowledge," in *Knowledge-Based Systems in Artificial Intelligence*, (R. Davis and D. B. Lenat, eds.), pp. 227-490, New York, NY: McGraw-Hill, 1982.
- [13] W. R. Swartout, "XPLAIN: a system for creating and explaining expert consulting programs," *Artificial Intelligence*, vol. 21, pp. 285-325, September 1983.
- [14] W. J. Clancey, "The epistemology of a rule-based expert system—a framework for explanation," *Artificial Intelligence*, vol. 20, pp. 215-251, May 1983.
- [15] W. van Meile, E. H. Shortliffe, and B. G. Buchanan, "EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems," in *Rule-Based Expert Systems*, (B. G. Buchanan and E. H. Shortliffe, eds.), pp. 302-313, Reading, MA: Addison-Wesley, 1984.
- [16] R. S. Patil, P. Szolovits, and W. B. Schwartz, "Causal understanding of patient illness in medical diagnosis," in *Readings in Medical Artificial Intelligence*, (W. J. Clancey and E. H. Shortliffe, eds.), pp. 339-360, Reading, MA: Addison-Wesley, 1984.
- [17] P. K. Fink, J. C. Lusth, and J. W. Duran, "A general expert system design for diagnostic problem solving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, pp. 553-560, September 1985.
- [18] J. S. Brown, R. R. Burton, and J. de Kleer, "Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III," in *Intelligent Tutoring Systems*, (D. Sleeman and J. S. Brown, eds.), pp. 227-282, London: Academic Press, 1982.
- [19] J. McDermott, "R1: A rule-based configurator of computer systems," *Artificial Intelligence*, vol. 19, pp. 39-88, September 1982.
- [20] W. J. Clancey, "Heuristic classification," *Artificial Intelligence*, vol. 27, pp. 289-350, December 1985.

REFERENCES

- [21] T. Bylander and B. Chandrasekaran, "Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition," *International Journal of Man-Machine Studies*, vol. 26, pp. 231-243, 1987.
- [22] T. Bylander and S. Mittal, "CSRL: A language for classificatory problem solving and uncertainty handling," *AI Magazine*, vol. 7, pp. 66-77, August 1986.
- [23] S. Mittal, B. Chandrasekaran, and J. Sticklen, "PATREC: A knowledge-directed data base for a diagnostic expert system," *IEEE Computer Special Issue*, vol. 17, pp. 51-58, September 1984.
- [24] C. Engelman, J. K. Millen, and E. A. Scarl, "KNOBS: An integrated ai interactive planning architecture," DSR 83-162, The Mitre Corporation, Bedford, MA, 1983.
- [25] D. C. Brown and B. Chandrasekaran, "Knowledge and control for a mechanical design expert system," *IEEE Computer*, vol. 19, pp. 92-100, July 1986.
- [26] J. W. Smith, J. R. Svrbely, C. A. Evans, P. Strohm, J. R. Josephson, and M. Tanner, "Red: A red-cell antibody identification expert module," *Journal of Medical Systems*, vol. 9, pp. 121-138, June 1985.
- [27] M. C. Tanner and J. R. Josephson, "Abductive justification," Technical Report, Laboratory for Artificial Intelligence Research, Ohio State Univ., Columbus, OH, March 1988.

Appendix D

Generic Tasks As Building Blocks for Knowledge-Based Systems: Diagnosis and Design

Generic Tasks As Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples¹

B. Chandrasekaran

**Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210**

June 28, 1988

¹To appear in *Knowledge Engineering Review*.

List of Figures

Figure 1:	<i>A Generic Task Architecture for Diagnosis With Compiled Knowledge</i>	9
Figure 2:	<i>Fragment of Fuel System classification tree. In this case, the hierarchy is largely of classes</i>	10
Figure 3:	<i>Skeleton specialist for BadFuel. The code specifies the location of BadFuel in the hierarchy,</i>	11
Figure 4:	<i>"Relevant" knowledge group of BadFuel. The ASK arguments are questions to the user, but</i>	12
Figure 5:	<i>"Summary" knowledge group of BadFuel.</i>	13
Figure 6:	<i>Establish procedure of BadFuel. If it has been evaluated not to be relevant, that sets the</i>	14
Figure 7:	<i>Example refine procedure. This specifies the control behavior for exploring the successors of</i>	14
Figure 8:	<i>An air-cylinder</i>	20
Figure 9:	<i>Specialist hierarchy for designing air cylinder. Design hierarchies may follow a</i>	21
Figure 10:	<i>Specialist "Head". The code specifies the location of the design specialist in the hierarchy,</i>	22
Figure 11:	<i>Plan "HeadDP1". The design plan specifies the specialist that uses it, what criteria should be</i>	22
Figure 12:	<i>Step "AirCavityID".</i>	23
Figure 13:	<i>Step "AirCavityID" continued.</i>	24

Abstract

The level of abstraction of much of the work in knowledge-based systems (the rule, frame, logic level) is too low to provide a rich enough vocabulary for knowledge and control. I provide an overview of a framework called the Generic Task approach that proposes that knowledge systems should be built out of building blocks, each of which is appropriate for a basic type of problem solving. Each generic task uses forms of knowledge and control strategies that are characteristic to it, and are in general conceptually closer to domain knowledge. This facilitates knowledge acquisition and can produce a more perspicuous explanation of problem solving. The relationship of the constructs at the generic task level to the rule-frame level is analogous to that between high level programming languages and assembly languages in computer science. I describe a set of generic tasks that have been found particularly useful in the constructing diagnostic, design and planning systems. In particular, I describe two tools, CSRL and DSPL, that are useful for building classification-based diagnostic systems and skeletal planning systems respectively. I describe a high level toolbox that is under construction called the Generic Task toolbox.

1 Need for Task-Specific Tools

The current generation of knowledge-based system (KBS) languages -- those that are based on rules, frames, or logic -- do not distinguish between different types of knowledge-based reasoning. For example, one would expect that the task of designing a car would require significantly different reasoning strategies than the task of diagnosing a malfunction in a car. However, these methodologies apply the same strategy (fire the rules whose conditions match, run resolution engine on all propositions etc.) to both design and diagnosis, as well as any other task. Because of this, it has been argued that these methodologies, although useful, are rather low level with respect to modeling the needed task-level behavior. In essence, these systems resemble an assembly language for writing KBS's. While they are obviously useful, clearly approaches that more directly address the higher level issues of knowledge-based reasoning are needed for the next generation of AI system development.

One example of a higher level approach is the *generic task* (GT) [9, 10, 12]. The aim here is to identify "building blocks" of reasoning strategies such that each of the types is both generic and widely useful as components of complex reasoning tasks. We have identified a number of such generic strategies, which together capture the functionality of a large portion of current expert systems. Each generic task² is characterized by:

1. The kinds of information it takes as input for the task and the information produced as a result of performing the task. This defines the functionality of the task.
2. A way to represent and organize the knowledge that is needed to perform the generic task. This includes a vocabulary of knowledge types, i.e., knowledge constructs that the representation language should have.
3. The process (algorithm, control, problem solving) that the task uses. This provides a vocabulary for inference and control for the task.

The GT framework proposes that, as each task and its associated structure is identified, languages

²Each GT is a *strategy* from the viewpoint of the problem which it is helping to solve. It is a *task* from the viewpoint of the functionality that is to be achieved by the GT. It is in this sense that a GT is both a strategy for a task and a task in itself. This distinction is in fact much more general: e.g., diagnosis is a strategy in making patients feel better (i.e., one way to organize therapeutic actions is to base them on causes, finding which is, in fact, diagnosis), but it is a task which many expert systems are designed to solve.

be developed that encode both the problem solving strategy and knowledge that is appropriate for solving problems of that type. These languages facilitate KBS development by giving the knowledge engineer access to tools which work closer to the level of the problem, not the level of the implementation language such as rules or frames. However, for nontrivial problems it may be necessary to decompose it into subproblems such that each matches the functionality of some GT. For example, we will show how certain kinds of diagnostic problems can be decomposed into a number of GT's. This way of building complex KBS's also means that knowledge engineering environments should provide a *toolset* rather than a single tool.

This style of supporting higher level generic computational activities with appropriate constructs is well-known in computer science in general; high-level programming languages are attempts to provide the programmer with constructs for a variety of common functions. It is items 2 and 3 above, viz., *knowledge and inference*, that make the above specification different from the standard high-level language constructs in computer science and make it particularly appropriate for knowledge-based problem solving.

2 Some Generic Tasks and Their Specifications

While the approach has relevance to AI in general, as a practical matter, our work has concentrated on information-processing strategies useful for building systems for knowledge-rich problem solving, or so-called expert systems. Such systems emphasize the role of large amounts of domain knowledge compiled for specific problem solving tasks that characterize routine human expert behavior. Without intending any kind of completeness, we list here some of the tasks that we have found to be very useful in building practical knowledge-based systems. As we will see, a variety of diagnostic and routine design and planning problems come under this category. The description of the tasks in this section is necessarily cryptic and somewhat oversimplified, and is provided mainly for a quick overview and comparison. Hierarchical classification, hypothesis matching, and plan selection and refinement are described in some detail in later sections of the paper, and abductive assembly described in somewhat less detail. Please see the citations for details on the rest of the generic tasks.

Each description is organized by the function of the task, the tool in our GT toolset for the task, the knowledge and inference types that the tool supports, and other relevant annotations. In each case, the GT tool commits itself to *one* way of achieving the functionality. Also, in each case the control behavior is the default behavior, and should be thought of as describing a family of control types.

2.a Hierarchical Classification

Task specification

Input: given a situation description of in terms of features. Output: classify it, as specifically as possible, in a classification hierarchy. (Multiple classifications, where different classes characterize different parts of the situation description, are also possible.)

GT tool

CSRL [6] (Conceptual Structures Representation Language).

How CSRL works:

Forms of Knowledge

Classification hierarchy, access to knowledge that produce information about how well the data match the classificatory concepts (see Hypothesis Matching, below)

Inference and Control

(Simplified) *Establish* nodes, if successful *refine* the concept by considering children, if unsuccessful, *reject* node, and *prune* subtree.

Example use

Medical diagnosis can often be viewed as partly a classification problem [22]. Problems may be classified into types which may then suggest methods of solutions. The diagnostic portion of MYCIN [35] (see Clancey [18]) and PROSPECTOR [19] can be viewed as classification problem solving.

2.b Hypothesis Matching

Task specification

Input: given a concept (a hypothesis) and set of data (features) that describe the problem state, Output: decide how well the concept matches the situation. The task is a form of *recognition* [26].

GT Tool

HYPER (HYPothesis matchER) [24]

How HYPER works:

Forms of Knowledge

An hierarchy of evidence abstractions, lowest level at the level of data and the highest level at the level of the concept. Each node abstracts from its children into a higher level feature. In the particular version considered in our work, the abstractions are qualitative degrees of confidence. (See detailed description of HYPER, Section 3.c.3.)

Inference and Control

At each level a degree of confidence in the presence of the feature is computed from the features that constitute evidence for it, and this is performed recursively until a degree of confidence for the concept is computed. The basic theory is that recognition of a complex concept is performed by hierarchically computing intermediate abstractions from raw data.

Example use

Samuel's *signature tables* perform this kind of abstraction. Many forms of *recognition* can be performed by means of this strategy. For example, the concept may be a disease and the data may be patient data relevant to the disease, and we wish to know what the likelihood of the disease is. Bylander et al [8] discuss a class of strategies called *structured matching* (of which the HYPER strategy is a particular example) and show how ubiquitous it is in knowledge-based reasoning.

2.c Knowledge-Directed Information Passing

Task specification

Input: Given attributes of some data entities, Output: determine the attributes of other data of interest, but not directly known, but can be inferred from the available data.

GT tool

IDABLE (Intelligent Data Base Language).

How IDABLE works:

Forms of Knowledge

Data concepts organized as a frame hierarchy of types and subtypes of data objects in the domain. Data attributes are slots with default values, default methods of computation of values, or explicit procedural attachments which specify how to compute data attribute from related data.

Inference and Control

Data queries result in the default value being chosen if no information is available, or the knowledge-intensive procedures to be invoked for inference. The inference procedures themselves can be inherited from parent concepts as needed.

Example use

A diagnostic system may use a knowledge-directed database of this type for converting from sensor or chart values into data of direct relevance to diagnosis. Clancey's data abstraction component of heuristic classification [18] can be achieved by this functionality.

2.d Synthesis by Plan Selection and Refinement

Task specification

Designing an object (device, program, plan) by hierarchical planning. Input: given specifications of the object to be designed Output: generate design of an object (device, program, plan) meeting the specifications.

GT tool

DSPL [3, 4] (Design Specialists and Plans Language).

How DSPL works:

Forms of Knowledge

Hierarchical structure of the object to be designed known in advance (making it routine design). For each node in the hierarchy, precompiled design plans are known for making design choices. Failure handling knowledge available, and some parts of the plans are constraint satisfaction knowledge, i.e., knowledge of constraints to be met by the design parameters.

Inference and Control

Top down control is typically used. Design plans are chosen, choices made at that level of abstraction, and design refined by calling on plans for the children (i.e., components). Plan failures are passed up until failure handling knowledge is available to fix the design or choose alternate plan.

Example systems

The task performed by the expert system MOLGEN [20] and R1 [29] can be viewed in this way. A variant of this task has also been called *skeletal planning* in the literature.

2.e Abductive Hypothesis Assembly

Task specification

Input: Given a situation description and a set of hypotheses each explaining some aspects of the situation and each with some plausibility value Output: Construct a composite hypothesis that is the best explanation of the situation, i.e., explains all the data parsimoniously and as well as possible.

GT tool

PEIRCE (named after C. B. Peirce, who first described the form of inference known as abduction) [34].

How PEIRCE works:

Forms of Knowledge

Causal or other relationships between hypotheses (such as incompatibility, special case of), relative significance of data describing the situation.

Inference and Control

Assembly and criticism alternate. At each stage during assembly the problem solving is driven an attempt to explain the most significant datum remaining unexplained. The best hypothesis that offers to explain it is added to the composite hypothesis. During criticism, explanatorily superfluous parts are removed. This loops until all the data are explained or no hypotheses are left.

Example use

This task is a subtask in diagnostic reasoning as well as in theory formation in science. Portions of INTERNIST [30] and DENDRAL [5] systems perform this task.

2.f Some Implications of GT's

The following general points about GT's are worth noting at this point.

1. As mentioned, a number of well-known expert systems can be thought of as decomposable into subtasks, each of which corresponds to one of the above tasks. R1 performs a simplified type of plan selection and refinement. Mycin performs classification and data abstraction (one of the capabilities of our knowledge-directed information passing) in the diagnostic part, and plan selection (in the therapy part). Additional examples can be given. Note, however, that in all these instances, we are only pointing out that these systems perform these tasks, but not necessarily in the manner that we propose that the tasks be performed. Our claim will be that once we understand the knowledge requirements and the inference strategies for each of the tasks, we can use methods that are more natural for the tasks.
2. We have mentioned diagnosis a number of times in the above description as a problem that uses one or more of the above generic tasks, e.g., classification and hypothesis assembly. Note that, correspondingly, we do not have a generic task called diagnosis in the above list. The reason for this is that, while diagnosis is "generic" in the sense that the problem occurs in a number of domains and there are similarities in the methods that are domain-independent, it is still a "compound" task in the sense that a number of distinct types of

knowledge and inferences are used in the process of doing diagnosis. Thus the above list of tasks can be used as natural building blocks for putting together a diagnostic problem solver. (We shall soon describe how this can be done.) This illustrates an additional constraint in our sense of generic task. The task needs to have a coherence and simplicity to it in that it ought to be characterizable by a simple type of knowledge and a family of inference types. This is what makes them "building blocks."

3. Functional modularity is an important consequence of this point of view. As we have shown in a number of papers [7, 9, 10, 12, 15] this functional modularity makes system building and debugging easier, and the task-specific knowledge and control constructs help in knowledge acquisition and explanation.
4. The above list is not meant to be a complete list of generic tasks useful in knowledge-based problem solving. In fact, quite a large part of AI -- from weak methods to qualitative simulation to scripts and plans -- can be thought of as attempts to identify interesting problems, the kind of knowledge required for it and the kinds of inferences useful to perform them. Thus, in qualitative reasoning, the generic problem considered is one where given the structure of a system, the task is to derive the system's behavior in a qualitative way. The research program then identifies the knowledge and inference for the task. In the appropriate context, each of them can be thought of as possible generic tasks. Our goal in the development of the generic task theory and the tool set has been to produce a methodology and a technology that helps in the analysis, design, construction and debugging of practical knowledge systems and thus we concentrated on the generic tasks that we felt would be most useful at this stage in the development of the technology. Research is underway in our Laboratory on other such generic tasks, which would cover phenomena in deep models such as structure to behavior reasoning and functional reasoning.

In the next sections I will describe how certain kinds of diagnostic and design problems can be built in the GT framework.

3 Diagnostic Reasoning

3.a Information Processing in Diagnosis

Abstractly, diagnosis is the problem of finding a cause or set of causes that "best explain" a set of observations of a system, some of them indicating behavioral abnormality. In most nontrivial cases, the process is a form of *abductive* reasoning, i.e., the diagnostic conclusion is not deductively provable, but is the hypothesis that makes best sense taking all the information into account.

We can identify a class of systems that we call *compiled knowledge systems* for diagnosis. These systems have knowledge that is needed for diagnosis precompiled. This knowledge, at a minimum, would include:

- knowledge of possible causes in terms of which the diagnostic answer will need to be given.
- knowledge that helps map from observations to possible causes, i.e., evaluate how likely a given cause or subset of causes might be given the set of observations.

Many of the well-known diagnostic expert systems, e.g., Mycin [35], Internist [30], MDX [39], set-covering and Bayesian diagnostic systems have this knowledge compiled in the knowledge base. Where these systems differ is in the *form* this knowledge takes, in the way the actual inference processes work, and also in the control of reasoning. Such compiled knowledge systems concentrate their problem solving behavior only on the specific diagnostic problem at hand, rather than in activities that produce the needed knowledge. These systems ought to be contrasted with diagnostic systems which do not have the needed diagnostic knowledge in a readily usable form (or the diagnostic knowledge is incomplete), but must acquire them by other kinds of problem solving, e.g., by *deriving* it from structural models of the device under diagnosis, or from analysis of past diagnostic cases involving the device. See [13] for a discussion of the general issues surrounding the use of deep models and [38] for a discussion of how such model-based reasoning and compiled reasoning can be interlaced. The diagnostic architecture that I will be discussing in the following pages is a compiled knowledge architecture.

Diagnosis can be computationally complex: even with compiled knowledge, all subsets of hypotheses may in principle need to be evaluated and compared. This is mainly due to two reasons: one, hypotheses may interact, i.e., two hypotheses together may account for more or less observations than the union of the sets of observations that they explain individually; and, two different subsets may explain the same sets of data and principles of parsimony will need to be brought in to choose the better explanation. All diagnostic systems, be they formal, such as set-covering and Bayesian approaches, or "heuristic", such as Mycin, either squarely face this problem and end up with computationally intractable algorithms, or make more or less realistic assumptions about the domain that help them cut down the exhaustive search through the space of hypotheses combinations. (For example, its domain is such that Mycin can implicitly make assumptions of no interaction between diagnostic hypotheses.)

The GT architecture that we will propose shortly broadly decomposes the problem into a *classificatory* one, which generates highly plausible diagnostic hypotheses, which are then used by an *abductive assembly* component to produce the best composite hypothesis for the problem. The overall *decomposition* above brings significant computational advantages, since the assembly process now only needs to work with a much smaller number of initial hypotheses, with the option to seek out less plausible hypotheses as needed for explanatory completeness. This, in conjunction with the computational efficiencies that the proposed architectures for classification and abductive assembly individually possess, makes the above architecture computationally attractive whenever knowledge is available in appropriate forms: e.g., hierarchies for classification and explicit knowledge about causal and logical interactions among diagnostic hypotheses for the abductive assembly component.

3.b A GT Architecture for Diagnosis

The architecture has four components: *hierarchical classification*, *hypothesis matchers*, *abductive assembly*, and *knowledge-directed data abstraction and inference*. The hierarchical classifier navigates the space of malfunctions organized as one or more hierarchies. The hierarchical organization permits a

quick determination of the plausible hypotheses with minimal search through the space of all possible hypotheses. The result of the classification process is a small set of highly plausible hypotheses.

The classifier itself needs a mechanism to evaluate the degree of plausibility of each of the hypotheses. The knowledge necessary to evaluate the plausibility of a classificatory hypothesis can be localized to each hypothesis in the context of the hierarchy. This can be done by a *hypothesis matching* component which evaluates any given hypothesis in the classification hierarchy by matching the data with expectations for the concept and which outputs a qualitative degree of confidence in the hypothesis (and the observations the hypothesis can explain). Thus the classifier, in conjunction with the hypothesis matchers for each of the concepts, can output plausible diagnostic hypotheses with the data it can explain attached to each of the hypotheses.

The output of the classifier goes to an *abductive hypothesis assembly* component, which puts together a subset of these hypotheses as a composite hypothesis that best explains the data. This process must consider the interactions that occur among the causes that correspond to the hypotheses in order to ensure internal coherence among combinations of hypotheses. The knowledge concerning hypothesis interactions can be explicitly represented for each hypothesis, thus being consulted only if that hypothesis is included in the composite hypothesis. This process must also assemble a diagnosis which meets various criteria of parsimony, completeness and plausibility. In the more complex uses of this architecture, the classification hierarchy may be asked to refine originally less plausible hypotheses if the explanatory power of the best composite hypothesis so far assembled is insufficient to cover all the observations that need explanation.

In many diagnostic problems the level of abstraction of the data which are available may be different from that required for the concept matcher, or additional inferences from available data may be needed to generate data that the diagnostic concept matcher can recognize as relevant. A necessary addition to this architecture is a database which uses domain knowledge to make the inferences and abstractions. In the proposed architecture, the hypothesis matcher can communicate with a system for *data retrieval/abstraction/inference*, whose task is a form of *knowledge-directed information passing* component which can convert data at to low a level of abstraction into diagnostically significant data.

The important point is that each of the modules above is generic:

- Each is a strategy independent of diagnosis and can be used in a number of other high level tasks. The abductive assembler, e.g., can just easily accept input from a plan recognizer so that it assembles a best explanation of various sightings in a battle situation. The data abstractor can be just easily used by a therapy planner, and so on.
- Each strategy, as we shall see, uses characteristic knowledge and inference, making it possible to focus the problem solving effort in a manner appropriate for the task.

The functionality of Clancey's *heuristic classification*, consisting of the subtasks of *data abstraction*, *heuristic matching*, and *refinement* can be achieved by three modules in our architecture: the hypothesis-matcher performs the heuristic matching task, the database component performs, among others, the data abstraction task, and the classifier performs the refinement task. The architecture has the additional capability of handling multiple malfunctions because of the abductive assembly component.

In sum the task of diagnosis can often be handled by the building blocks in the architecture diagramed in Figure 1. Thus, if the appropriate knowledge is available, a compiled knowledge diagnostic system can be built using the CSRL, HYPER, PEIRCE and IDABLE tools.

Gomez and Chandrasekaran [22] pointed out the importance of classification for diagnosis and Mittal [31] described the architecture of MDX, which included all the components except abductive assembly. Josephson et al [27] added abductive assembly as part of a general architecture for abduction.

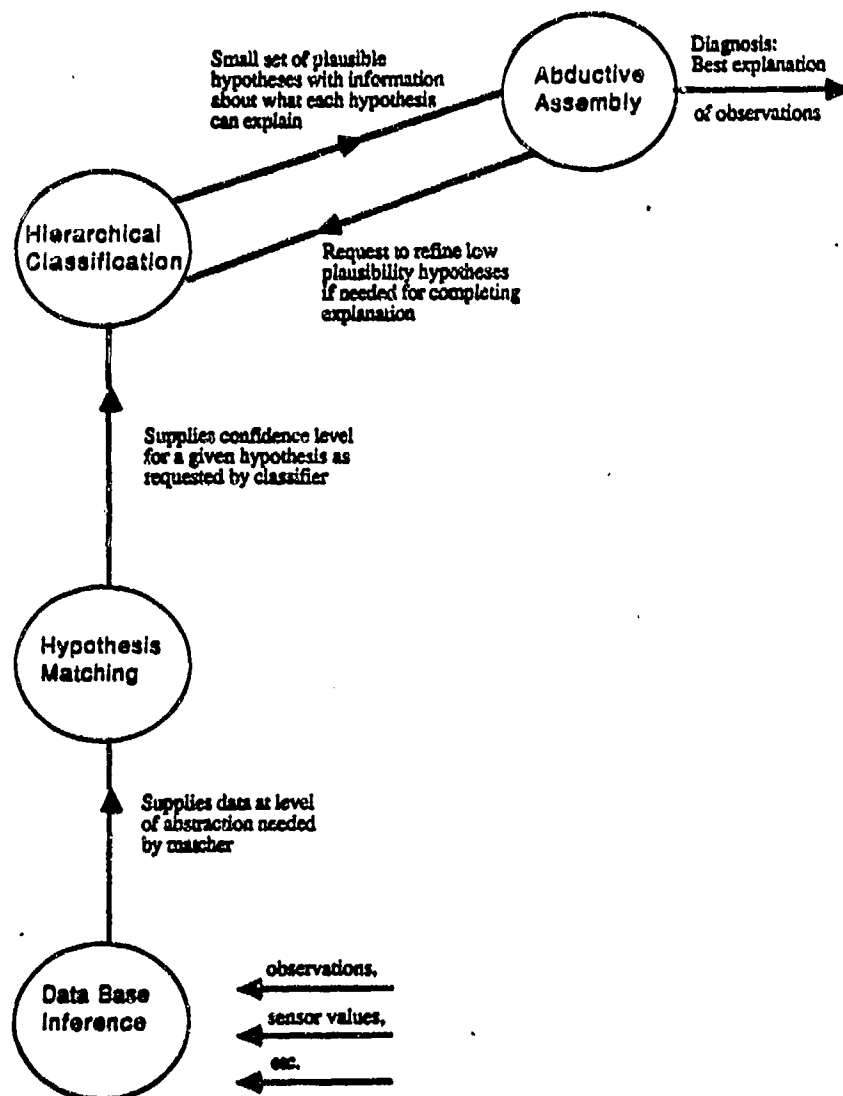


Figure 1: A Generic Task Architecture for Diagnosis With Compiled Knowledge

Describing how all the components of the diagnostic architecture can be built and integrated requires much more space than we have available. We describe in some detail the use of CSRL (the version to be described has HYPER embedded in it) for classification and hypothesis matching, and describe the assembly process in less detail. References [25] and [34] give further information on the tools Idable and Peirce.

3.c Classificatory Problem Solving and CSRL³

³Much of the material in this section is from B. Chandrasekaran and William F. Punch III, "Hierarchical classification: its usefulness for diagnosis and sensor validation," invited talk at the Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program, March 9-11, 1987.

3.c.1 Classificatory Hierarchies

Hierarchical classification (HC) is a particular method of performing the classification task. HC requires the availability of a classification hierarchy that organizes the classificatory hypotheses. Medical diagnosis, e.g., uses disease hierarchies, and in many engineering domains, malfunction hierarchies are quite common.

In Section 2, we gave a characterization of hierarchical classification. Figure 2 illustrates a fragment of a tree from a hierarchical classification system for the diagnosis of Fuel System malfunctions in a car engine.

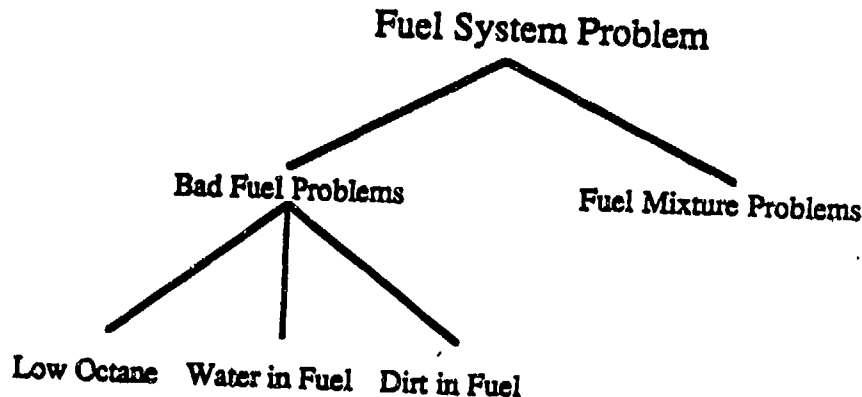


Figure 2: *Fragment of Fuel System classification tree. In this case, the hierarchy is largely of classes and subclasses of "causes." In other cases, the subclasses may be subfunctions, or physical parts.*

Note that as the hierarchy is traversed from the top down, the categories (or in this particular case, hypotheses about the failure of the fuel system) become more specific. Thus the children of the hypothesis Bad Fuel Problems can be broken into more specific hypotheses of Low Octane, Water in Fuel and Dirt in Fuel.

Each node in the hierarchy is responsible for calculating the "degree of fit" or *confidence value* of the hypotheses that the node represents. For example, the Bad Fuel Problems node is responsible for determining if there is a bad fuel problem and the degree of confidence it has in that decision. Each node can be thought of as a "specialist" in determining if the hypothesis it represents is present. To create each specialist, knowledge must be provided to make this confidence value decision. The general idea is that each specialist specifies a list of *features* that are important in determining whether the hypothesis it represents is present and a list of *patterns* that map combinations of features to confidence values. In the Fuel System Problems specialist, such features might include gas mileage problems, poor performance, difficulty in starting the engine etc. One pattern might be that if all the features are present, then the Fuel System Problems hypothesis is likely.

3.c.2 The Control Strategy of Hierarchical Classification

Given that the knowledge of the system is organized as a set of specialists in a hierarchy, how can the hierarchy be *efficiently* traversed? This process is primarily accomplished through a type of hypothesis refinement called *establish-refine*. Simply put, a specialist that *establishes* its hypothesis (has a high confidence value) *refines* itself by activating its more detailed sub-specialists. A hypothesis that is ruled out or rejected its hypothesis (has a low confidence value) is not refined, thus effectively pruning the subtree below it. The reason for this becomes obvious when one thinks again of how the specialists are organized. The subhypotheses of Fuel System Problems, for example, are simply more detailed hypotheses. If there is no evidence for Fuel System Problems (it is ruled out), then there is no point in examining more detailed hypothesis about failures of the fuel system.

The process of establish-refine continues until no more refinements can take place. This can occur either by reaching the tip level hypotheses of the hierarchy or by ruling out mid-hierarchy hypotheses.

3.c.3 CSRL, a Language Tool for Hierarchical Classification Systems

CSRL (Conceptual Structure Representation Language) [6] is a language for writing hierarchical classification expert systems. The current version of CSRL is really a mixture of the shells of both hierarchical classification and hypothesis matching. A new version of the hypothesis matcher shell is available as HYPER. In this section, we will describe the older form of CSRL.

CSRL allows a knowledge engineer to do three things:

1. Create a hierarchy of malfunction hypotheses in a particular domain.
2. Encode the pattern matching knowledge for each hypothesis into a specialist.
3. Control the process of establish-refine problem solving.

Encoding the Hierarchy of Malfunctions

In CSRL, a hierarchical classification system is implemented by individually defining a specialist for each malfunction hypothesis. The super- and sub-specialists of a specialist are declared within the definition. Figure 3 is a skeleton of a specialist definition for the Bad Fuel node from Figure 2. The declare section specifies its relationships to other specialists. The other sections of the specialist will be examined later.

```
(SPECIALIST BadFuel
  (DECLARE (SUPERSPECIALIST FuelSystem)
    (SUBSPECIALIST LowOctane WaterInFuel
      DirtInFuel))
  (KGS ...)
  (MESSAGES ...))
```

Figure 3: *Skeleton specialist for BadFuel. The code specifies the location of BadFuel in the hierarchy, points to the knowledge groups that contain information about how to establish or reject the concept, and contains messages that specify control behavior.*

Designing a classification hierarchy is an important part of building a CSRL expert system, but the exact structure of the final system is a pragmatic decision rather than a search for the perfect hierarchy. The main criterion for evaluating a classification hierarchy is whether enough evidence is normally available to make confident decisions. To decompose a specialist into its subspecialists, the simplest method is to ask the domain expert what subhypotheses should be considered next. The subhypotheses should be subtypes of the specialist's hypothesis, and will usually differ from one another based on a single attribute (e.g., location, cause).

For the diagnosis problem the criteria for forming classification hierarchies are discussed, with examples from the medical domain, by [7], and in the engineering domain by [32]. The hierarchy may mix function-subfunction and part-subpart views depending upon the way diagnostic reasoning actually works in the domain. Multiple hierarchies of the same domain, each from a different perspective, are also useful for some domains: the MDX-2 system of Sticklen [39] uses such multiple hierarchies.

Encoding Pattern-Match Knowledge

The knowledge groups in the kgs section contain knowledge that matches the features of a specialist against the case data. Each knowledge group is used to determine a confidence value for some subset of features used by the specialist. As such, a knowledge group becomes an abstraction of evidence, representing an *evidential abstraction* of a particular set of features important to establishing the specialist. A knowledge group is implemented as a cluster of production rules that maps the values of a list of expressions (boolean and arithmetic operations on data, values of other knowledge groups) to some conclusion on a discrete, symbolic scale.

As an example, Figure 4 is the relevant knowledge group of the BadFuel specialist mentioned above. It determines whether the symptoms of the automobile are consistent with bad fuel problems. The expressions in the MATCH part queries the user (who acts as the database for this case) concerning whether the car is slow to respond, starts hard, has knocking or pinging sounds, or has the problem when accelerating. ASKYNU? is a LISP function which asks the user for a Y, N, or U (unknown) answer from the user, and translates the answer into T, F, or U, the values of CSRL's three-valued logic (Note that any LISP function may be used here). The results of the MATCH expressions are then compared to a condition list in the WITH part of the knowledge group. For example, the first pattern "T ? ?" in the figure tests whether the first match expression (ASKYNU? "Is the car slow to respond") is true (the ? means doesn't matter). If so, then -3 becomes the value of the knowledge group⁴. Otherwise, subsequent patterns "? T ?" or "? ? T" are evaluated. The value of the knowledge group will be 1 if no rule matches. This knowledge group encodes the following matching knowledge:

If the car is slow to respond or if the car starts hard, then BadFuel is not relevant in this case. Otherwise, if there are knocking or pinging sounds and if the problem occurs while accelerating, then BadFuel is highly relevant. In all other cases, BadFuel is only mildly relevant.

(RELEVANT TABLE

(MATCH

(ASKYNU? "Is the car slow to respond")

(ASKYNU? "Does the car start hard")

(AND (ASKYNU? "Do you hear knocking or
pinging sounds")

(ASKYNU? "Does the problem occur while
accelerating"))

WITH (IF T ? ?

THEN -3

ELSEIF ? T ?

THEN -3

ELSEIF ? ? T

THEN 3

ELSE 1)))

Figure 4: "Relevant" knowledge group of BadFuel. The ASK arguments are questions to the user, but they can also be queries to the database. The argument of WITH specify truth tables in the knowledge group.

Figure 5 is the summary knowledge group of BadFuel. Its MATCH expressions are the values of the relevant and gas knowledge group (the latter queries the user about the temporal relationship between the onset of the problem and when gas was last bought). In this case, if the value of the

⁴In this case, the values assigned are on a discrete scale from -3 to 3, -3 representing ruled-out and 3 representing confirmed.

relevant knowledge group is 3 and the value of the gas knowledge group is greater than or equal to 0, then the value of the summary knowledge group (and consequently the confidence value of BadFuel) is 3, indicating that a bad fuel problem is very likely.

```
(SUMMARY TABLE
(MATCH RELEVANT gas
 WITH (IF 3 (GE 0)
 THEN 3
 ELSEIF 1 (GE 0)
 THEN 2
 ELSEIF ? (LT 0)
 THEN -3)))
```

Figure 5: "Summary" knowledge group of BadFuel.

This method of evidence combination allows the calculation of the confidence value to be hierarchically organized. That is, the results of any number of knowledge groups can be further abstracted by a knowledge group that can combine their values into a single confidence value.

As mentioned earlier the above pattern matching knowledge and problem solving structure is a generic task that we have identified as hypothesis matching, and a separate shell called HYPER is available to capture just this functionality.

The mapping from data to confidence in a concept is a form of probabilistic mapping. The symbolic degrees of confidence are qualitative measures of subjective likelihood. However, the way data combine to produce a confidence for a higher level feature is *not* modeled by any normative calculus, be it Bayesian or one based on fuzzy sets, but *directly* obtained from the domain expertise localized to that particular context. There are important issues of how this view of handling uncertainty differs from the more traditional formal methods for which we refer the reader to [11, 6].

Encoding of Establish-Refine Strategy

The MESSAGES section of a specialist contains a list of message procedures which specify how the specialist will respond to different messages from its superspecialist. ESTABLISH and REFINE are the predefined messages in CSRL though others may be created by the user. The establish message procedure of a specialist determines the *confidence value* (i.e., the degree of fit) of the specialist's hypothesis. Figure 6 illustrates the establish message procedure of the BadFuel specialist. relevant and summary are names of knowledge groups of BadFuel (see previous section). SELF is a keyword which refers to the name of the specialist. This procedure first tests the value of the relevant knowledge group. (If its knowledge group has not already been evaluated, it is automatically evaluated at this point.) If it is greater than or equal to 0, then BadFuel's confidence value is set to the value of the summary knowledge group, else it is set to the value of the relevant knowledge group. A value of +2 or +3 indicates that the specialist is established. In this case, the procedure corresponds to the following strategy.

First perform a preliminary check to make sure that BadFuel is a relevant hypothesis to hold. If it is not (the relevant knowledge group is less than 0), then set BadFuel's confidence value to the degree of relevance. Otherwise, perform more complicated reasoning (the summary knowledge group combines the values of other knowledge groups) to determine BadFuel's confidence value.

The refine message procedure determines what subspecialists should be invoked and the messages they are sent. Figure 7 shows a refine procedure which is a simplified version of the one that BadFuel uses. SUBSPECIALISTS is a keyword which refers to the subspecialists of the current specialist. The procedure calls each subspecialist with an ESTABLISH message. If the subspecialist establishes itself (+? tests if the confidence value is +2 or +3), then it is sent a REFINE message.

```

(ESTABLISH (IF (GE relevant 0)
  THEN (SETCONFIDENCE self summary)
  else (SETCONFIDENCE self relevant)))

```

Figure 6: *Establish procedure of BadFuel. If it has been evaluated not to be relevant, that sets the confidence value to be negative. If it is relevant then more complex matching is invoked to set the confidence value.*

```

(REFINE (FOR specialist IN subspecialists
  DO (CALL specialist WITH ESTABLISH)
  (IF (+? specialist)
    THEN (CALL specialist
      WITH REFINE))))

```

Figure 7: *Example refine procedure. This specifies the control behavior for exploring the successors of a classificatory hypothesis that has been established.*

3.c.4 The Computational Advantages of Hierarchical Classification

The major advantage of a hierarchical classification system is the organization of both the hierarchy of malfunctions and the knowledge groups within a specialist. This organization allows an efficient examination of the knowledge of the system based on need.

Consider again the hierarchy of Figure 2. The problem solving begins by evaluation of the specialist Fuel System Problems. If that specialist establishes, then the two sub-specialists Bad Fuel Problems and Fuel Mixture Problems are invoked. If however, the Bad Fuel Specialist does not establish, then none of its sub-specialists will be invoked. Thus, if a specialist rules out (i.e. does not establish), then none of the knowledge of the sub-specialists need be run.

The same is true of the knowledge groups in the specialist. Only that knowledge necessary to confirm or deny the knowledge group is run. If a row of the knowledge group matches, then none of the subsequent rows are evaluated. Again, this results in running only the knowledge necessary for the problem at hand.

Compare this with other hierarchical approaches to diagnosis. The *fault tree* is a sequence of causally related events that leads to an observable symptom in the system. Given an initial malfunction, all possible causal results of the event are traced out, terminating with the symptoms that would be observed by a human diagnostician. When applied to an entire system, the result is a network of events that represent all the causal relationships of the system's constituent parts. While useful in design tasks, application of fault trees to diagnosis has a number of problems.

1. The combinatorial fan-out from an initial event can be very large. This makes the job of creating and traversing the network difficult. Compare this with the abstraction of hypotheses in hierarchical classification systems. Each node in the hierarchy represents a malfunction hypothesis that is listed in more detail through its sub-specialists. If many sub-specialists occur in the hierarchical decomposition of the domain, more levels of

abstraction can be introduced to limit the fan-out. Such abstraction does not exist in fault tree representations.

2. Fault trees make no attempt to limit the number of nodes of the network that must be evaluated. Given a significant event, all possibilities are examined. However, hierarchical classifiers make use of the abstraction of malfunction hypotheses to limit the number of nodes that must be examined based on the data of the case.

The issues of control and communication in hierarchical classification can be more complex than our description in this paper. Gomez and Chandrasekaran [22] describe the use of blackboards in exchanging information between different portions of the hierarchy. Sticklen, et al [38] describe the more complex control issues that need to be faced in some situations, and Sticklen [39] describes the use of multiple hierarchies in classification.

3.d Hypothesis Assembly in Diagnosis

In typical diagnosis problems, the available data cannot always be explained by one malfunction hypothesis, but may require several hypotheses which must be combined in order to account for the observations. As the space of hypotheses grow in size, the problem of finding the best combination of hypotheses which apply to a particular situation becomes exponentially more difficult. Hierarchical classification can trim down the space of applicable hypotheses tremendously, yet it may still be necessary to find a subset of the hierarchy's plausible candidates which gives the simplest and best diagnosis.

For example, suppose that the Auto-Mech hierarchy were extended to classify malfunctions for other car subsystems, including the braking mechanism. When classifying a case which has data such as "The car won't start when cold," "The engine runs roughly," "The brakes are hard to push," and "The car doesn't stop quickly," several diagnostic hypotheses may be found to be applicable, such as 'water in fuel' and 'loss of brake fluid', among others. In this case, the classification hierarchy has trimmed the set of all fuel and brake system diagnostic hypotheses down to a handful of highly relevant ones. However, the classification mechanism as such is incapable of determining whether this subset of relevant hypotheses accounts for all of the data. Furthermore, some of these hypotheses may be inconsistent or superfluous with respect to each other. Therefore, it is necessary to invoke a process of hypothesis assembly to complete the diagnosis by assembling a parsimonious subset of these hypotheses which gives the best explanation. In this case, the final subset of hypotheses could be 'water in fuel' and 'loss of brake fluid', since both are highly plausible, between the two of them they account for all the data, and neither is inconsistent or superfluous with regard to the other.

To carry this example further, it is possible that the hypothesis assembler may uncover relevant relationships which are only implicitly represented in the classification hierarchy. For example, in some models of cars the vacuum generated by airflow through the engine is diverted to assist the braking mechanism. Thus a punctured vacuum hose reduces the airflow through the carburetor, causing the engine to run roughly, and disabling the assistance to the brakes. When running the Auto-Mech hierarchy on the previous case, for an appropriate model of car, the following hypotheses may be found to be relevant: 'engine vacuum hose punctured' and 'brake vacuum assist inoperational'. Both of these hypotheses need to be present in the classification hierarchy, because the first refines the 'engine problem' hypothesis, whereas the second refines the 'brake system problem' hypothesis. In this case, however, it is clear that the second problem is causally related to the first. Therefore, when the hypothesis assembler is putting together the best explanation for hypothesis assembler is putting together the best explanation for the data, it will uncover that 'engine vacuum hose punctured' accounts for all the data that 'brake vacuum assist inoperational' accounts for, and propose 'engine vacuum hose punctured' as a one-hypothesis set which best explains the data.

Combining the process of hierarchical classification and hypothesis assembly in this way results in an efficient process for diagnosis. Classification without assembly cannot evaluate the final diagnosis in the case of multiple failures. Assembly without classification is an intractable problem in the general case. The issue of complexity in hypothesis assembly is particularly noticeable in the medical domain, where physiological and anatomical subsystems interact in highly complex ways. When both generic tasks are combined, however, their interaction reduces the complexity of the diagnostic problem while maintaining the ability to find the best explanation when there is one.

Josephson, et al [27] is a good source of how the abductive assembly process works. Punch et al [34] describe the tool PEIRCE, which is intended to build abductive assembly systems.

3.e Applications of the Abductive Architecture

A number of diagnostic systems have been built using the hierarchical classification approach provided by the CSRL tool. This section enumerates some of these applications and their domains.

It should be noted that of the following systems, Auto-Mech is strictly a pedagogical system, the Nuclear Power and Chemical Engineering systems are initial explorations for yet to be developed systems and Red, WELDEX and ROMAD are being developed to be used in real world situations.

Auto-Mech [40]

Auto-Mech is an expert system which diagnoses fuel problems in automobile engines. The purpose of the fuel system is to deliver a mixture of fuel and air to the air cylinders of the engine. It can be divided into major subsystems (fuel delivery, air intake, carburetor, vacuum manifold) which correspond to initial hypotheses about fuel system faults.

Auto-Mech consists of 34 CSRL specialists in a hierarchy which varies from four to six levels deep. Before running, Auto-Mech collects some initial data from the user. This includes the major symptom that the user notices (such as stalling) and the situation when this occurs (e.g., accelerating and cold engine temperature). Any additional questions are asked while Auto-Mech's specialists are running. The diagnosis continues until the user is satisfied that the diagnosis is complete.

A major part of Auto-Mech's development was determining the assumptions that would be made about the design of the automobile engine and the data that the program would use. Different automobile engine designs have a significant effect on the hypotheses that are considered. A carbureted engine, for example, will have a different set of problems than a fuel injected engine (the former can have a broken carburetor). The data was assumed to come from commonly available resources. The variety of computer analysis information that is available to mechanics today was not considered in order to simplify building Auto-Mech.

Red [37]

Red is an expert system whose domain is red blood cell antibody identification. An everyday problem that a blood bank contends with is the selection of units of blood for transfusion during major surgery. The primary difficulty is that antibodies in the patient's blood may attack the transfused blood, rendering the new blood useless as well as presenting additional danger to the patient. Thus identifying the patient's antibodies and selecting blood which will not react with them is a critical task for nearly all red blood transfusions.

The Red expert system is composed of three major subsystems, one of which is implemented in CSRL. The non-CSRL subsystems are a data base which maintains and answers questions about reaction records (reactions of the patient's blood in selected blood samples under a variety of conditions), and a overview system, which assembles a composite hypothesis of the antibodies that would best explain the reaction record. (This assembly is itself a generic task called "abductive assembly" and a tool called PEIRCE can be used to build the assembly system.) CSRL is used to implement specialists

corresponding to the common blood antibodies and to each antibody subtype (different ways that the antibody can react).

The major function of the specialists is to rule out antibodies and their subtypes whenever possible, thus simplifying the job of the overview subsystem, and to assign confidence values, informing overview of which antibodies appear to be more plausible. The specialists query the data base for information about the lab test results and other patient information, and also tell the data base to perform certain operations on reaction records.

Complex Mechanical Systems

CSRL has been used in creating expert systems that do diagnosis of faults both in the domain of Nuclear Power Plants and in the domain of Chemical Engineering.

The Nuclear Power Industry must be very careful in the maintenance of running power plants since mistakes can prove costly not only in terms of power plant damage but also in terms of radiation leakage and broad environmental damage. Nuclear Power Plants are therefore heavily monitored in many areas, so heavily in fact that it is difficult (if not impossible) for the operator to maintain an understanding of just what exactly is going on. The Nuclear Power Plant expert system [23] is designed to take in large amounts of data and classify them into one of approximately 25 different failures. One advantage of the CSRL approach is that the operator can be informed of a high level view of the problem if no specific failure can be discovered.

The problems of the Chemical Engineering Plant are similar, but it does have a number of differences. While safety is also of concern, there is also the problem of product quality in a Chemical Engineering Plant. If a malfunction occurs that produces an unusable product, the operation must be brought quickly back into line or large amounts of material will be wasted. The Chemical Engineering expert system [36] does diagnosis of a typical reactor producing a solid product as a result of the reaction of liquid product and oxygen. It consists of approximately 30 specialists that represent hypotheses about failures of the various physical parts of the plant. In addition to data that monitors the state of the reactor, these specialists also use data about product quality to make the confidence value decision.

Other Real World Uses of CSRL

CSRL is being used to develop two commercial systems by the Knowledge Based Systems group at the Battelle Columbus Institute. WELDEX and ROMAD are diagnostic systems for, respectively, detecting welding defects and evaluating machinery. A brief description of WELDEX follows.

WELDEX identifies possible defects in a weld from radiographic data of the weld. Industry standards and regulations require careful inspection of the entire weld and a very high level of quality control. Thus for industries which rely on welding technology, such as the gas pipeline industry, radiograph inspection is a tedious, time-consuming, and expensive part of their operations.

This problem can be decomposed into two tasks: visual processing of the radiograph to extract relevant features of the weld, and mapping these visual features to the welding defects which give rise to them. WELDEX is intended to perform the second task. The current prototype consists of 25 CSRL specialists that are organized around different regions of the weld, taking advantage of the fact that each class of defects tends to occur in a particular region. The knowledge groups in these specialists concentrate on optical contrast, shape, size and location of the radiograph features. A customer version of WELDEX is currently being developed. Future work is anticipated on developing a visual processing system whose output would be processed by WELDEX, thus automating both parts of the radiograph inspection problem.

4 Routine Design and DSPL⁵

4.a Subprocesses in Design

Design is in general complex and, from the viewpoint of AI, a relatively poorly understood activity. For our purposes here, it is useful to think of design problem solving as having two sets of parts: those that "generate," i.e., propose designs or parts of designs and those that "test," i.e., analyze, critique and evaluate designs. Evaluating a design may involve problem solving behavior such as qualitative simulation (e.g., to see if the projection from the wheel will rub against the body during rotation) or quantitative analyses (e.g., finite element methods to evaluate stress in a design component to see if the maximum stress is below the specifications), but these processes are not specific to design as a problem solving activity. On the other hand, the processes that participate in the "generate" portion are specific to design. In [16] we have identified some of these generic processes:

- design problem decomposition,
- design plan instantiation and expansion,
- retrieval and modification of similar designs and,
- global satisfaction of constraint equations.

Among the above four processes, the first two are especially important for the discussion in this paper. In design by decomposition pre-stored domain knowledge is available which proposes possible decompositions of the design problem into specific subproblems, each hopefully of lesser complexity. In design by plan selection, instantiation and refinement, similarly a prestored *design plan* is available setting out a sequence of steps, some of them involving making some design commitments and others possibly involving calling other design plans, for solving the design problem at hand. The combination decomposition and design plan instantiation and refinement can lead to quite complex problem solving. In design from past cases, both the decomposition and design plan are implicitly available for a previous design case, but they typically call for further criticism and modification.

4.b Classes of Design

The framework suggests that design by decomposition (i.e., breaking problems into subproblems), by plan selection, and by plan synthesis (as a last resort) are the core processes in knowledge-based design. This suggests an informal classification of design problems based on the difficulty of these subtasks or processes.

4.b.1 Class 1 Design

This is open-ended "creative" design. Goals are ill-specified, and there is no storehouse of effective decompositions, not to speak of design plans for subproblems. Even when decomposition knowledge is available, most of the effort is in searching for potentially useful problem decompositions. For each potential subproblem, further work has to be done in evaluating if a design plan can be constructed. This design problem is not routine. The average designer in industry will rarely, if ever, do Class 1 design: such design leads to an invention or new products.

⁵Parts of this section taken from a number of joint papers by the author with D. C. Brown.

4.b.2 Class 2 Design

Class 2 design is characterized by powerful problem decompositions already available, but design plans for some of the component problems in need of *de novo* construction or substantial modification. Design of a new automobile, e.g., does not involve new discoveries about decomposition: the structure of the automobile has been fixed for quite a long time. On the other hand, several of the components in it constantly undergo major technological changes, and routine methods of design for some of them may no longer be applicable.

Complexity of failure analysis will also take a problem away from routine design. Even if design plans are available, if the problem solver has to engage in very complex problem solving procedures in order to decide how to backtrack, the advantage of routine design is reduced. In short, whenever substantial modifications of design plans for components are called for, or when synthesis in the design plan space is especially complicated, we have a Class 2 problem.

4.b.3 Class 3 Design

This is relatively routine design: effective problem decompositions are known, compiled design plans for the component problems are known, and actions to take on failure of design solutions are also explicitly known. There is very little complex auxiliary problem solving needed. In spite of all this simplicity, the design task itself is not trivial: complex backtracking can still take place. The design task is still too complex for simple algorithmic solutions or table look up.

Class 3 problems are routine design problems, but still requiring knowledge-based problem solving. The ensuing sections of this paper deal with an approach to building knowledge-based systems for routine design problems of this type. The processes described here can work in conjunction with auxiliary problem solvers of various types, but we do not discuss such additional problem solvers here. The examples used all assume that the information to be provided by the auxiliary design processes, e.g., design criticism, verification, and subproblem constraint generation, are all available in a compiled manner.

4.b.4 A Class 3 Product

In a large number of industries, products are tailored to the installation site while retaining the same structure and general properties. For example, an aircraft intended for accurate and reliable backward and forward movement of some component will need to be redesigned for every new customer in order to take into account the particular space into which it must fit or the intended operating temperatures and pressures. This is a design task, but a relatively unrewarding one, as the designer knows at each stage of the design what the options are and in which order to select them. Note that that doesn't mean that the designer knows the complete sequence of steps in time (i.e., the trace) in advance, as the designer has to be in the problem-solving situation before each decision can be made. There are just too many combinations of requirements and design situations to allow an algorithm to be written to do the job. This class of problems, while simple, is not by any means trivial: in fact, a typical class 3 problem involves more complex problem solving behavior at the design level than say is implicit in R1 [29].

DSPL is a language designed by D. C. Brown [3] which captures the problem decomposition knowledge in the form of a design hierarchy of *design specialists* and the planning knowledge of each specialist is in the form of *design plans*. The specialists also have a certain amount of compiled *failure handling knowledge*, i.e., knowledge to help them recover when any of the chosen plans fail to accomplish their mission.

The approach taken is to consider design knowledge to be in the form of active cooperating design specialists. These specialists are organized in a hierarchy that reflects the human designer's conceptual organization of the design activity. Specialists use their own local design knowledge, but can also use the specialists directly below them in the hierarchy. This use is controlled by plans embedded in every specialist. Each specialist is responsible for some portion of the design, while its plans represent alternative methods for designing that portion. Communication between specialists is in the form of messages that flow up and down the hierarchy between the specialists and between their local agents

(i.e., local design knowledge)⁶. Messages flowing up may indicate failure or success.

The domain chosen was that of designing an Air Cylinder that was used by a local company in many pieces of equipment but which needed to be designed again each time due to changing requirements, such as the air pressure and the length of the stroke. A system called AIR-CYL has been written that does the design given a set of requirements.

The rest of this section will describe DSPL using the example of AIR-CYL [4].

4.c DSPL : The Design Specialists and Plans Language

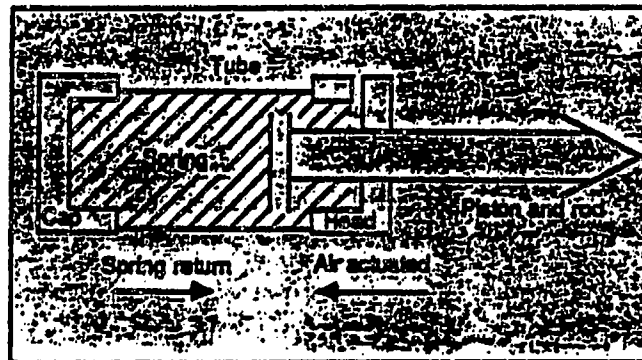


Figure 8: An air-cylinder

The air cylinder (AC) has about 15 parts, almost all of which are manufactured by the company according to their own designs, as their requirements are such that the components cannot be purchased. The AC is redesigned and changed slightly for applications with markedly different requirements. This characteristic makes it routine design in type. In operation, compressed air forces a piston back into a tube against a spring. Movement is limited by a bumper. The spring returns the piston, and the attached "load," to its original position when the air pressure drops.

The corresponding design specialist hierarchy is given in Figure 9. The expert system for design of air cylinders is organized as a hierarchy of such specialists.

DSPL provides a way of writing declarations of Specialists, Plans, Tasks, Steps, Constraints, Failure Handlers, Redesigners, Sponsors and Selectors, allowing the user to specify the knowledge contained in them. In the following section, we will address each of these declarations in turn.

⁶We use the term "agent" to denote any chunk of knowledge that performs some action with well-defined functionality to it. The term "specialist" refers to those agents which correspond to the nodes in the part-subpart hierarchy of the object under design. Each specialist consists of further agents which take care of the subtasks of the specialists. In DSPL, these agents come in a number of different types.

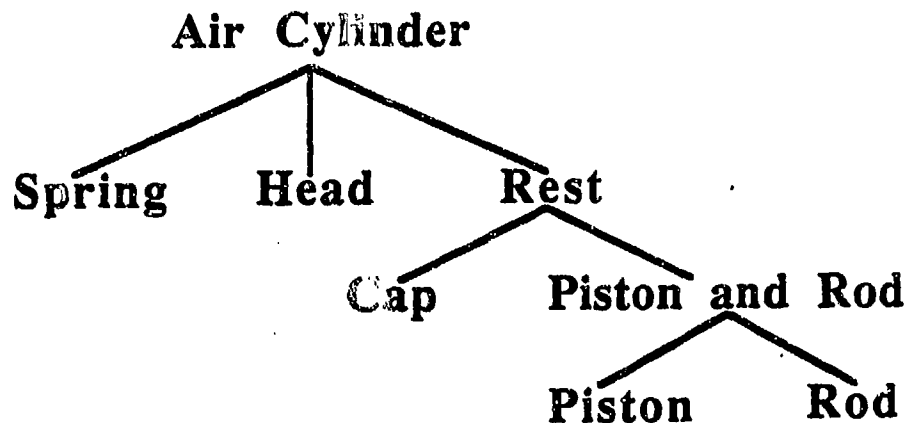


Figure 9: *Specialist hierarchy for designing air cylinder. Design hierarchies may follow a function-subfunction or a part-subpart or a combination thereof as principles for organizing the hierarchy.*

To build a Design Expert System (DES), the user declares in DSPL all the agents (i.e., active design knowledge) required, and then allows the underlying system to link them together after some checking. Once formed the DES can be invoked by requesting a design from the top-most specialist. The design then proceeds according to the specialist's plans. After a successful termination the design data-base contains the completed design. If failure occurs reasons are given. The DSPL system provides the underlying problem-solving control.

4.c.5 Design Agents Specialists

A Specialist is a design agent that will attempt to design a section of the component. The specialists chosen, their responsibilities, and their hierarchical organization will reflect the mechanical designer's underlying conceptual structure of the problem domain. Exactly what each specialist's responsibilities are depends on where in the hierarchy it is placed. Higher specialists have more general responsibilities. The top-most specialist is responsible for the whole design. A specialist lower down in the hierarchy will be making detailed decisions. Each specialist has the ability to make design decisions about the part, parts or function in which it specializes. Those decisions are made in the context of previous design decisions made by other specialists. A specialist can do its piece of design by itself, or can utilize the services of other specialists below it in the hierarchy. We refer to this cooperative design activity of the specialists as Design Refinement.

Every specialist also has some local design knowledge expressed in the form of constraints. These will be used to decide on the suitability of incoming requirements and data, and on the ultimate success of the specialist itself (i.e., the constraints capture those major things that must be true of the specialist's design before it can be considered to be successfully completed). Other constraints, embedded in the specialists plans, are used to check the correctness of intermediate design decisions. Still more constraints are present in the design data-base as general consistency checks. A typical specialist is shown in Figure 10.

The Selectors will be used to select from amongst the specialists plans. If no selector is specified a

```

(SPECIALIST
  (NAME      Head)
  (USED-BY AirCylinder)
  (USES      None)
  (DESIGN-PLANS      HeadDP1)
  (DESIGN-PLAN-SELECTOR Headdpselector)
  (ROUGH-DESIGN-PLANS HeadRDP1)
  (INITIAL-CONSTRAINTS None)
  (FINAL-CONSTRAINTS  None)

```

Figure 10: Specialist "Head". The code specifies the location of the design specialist in the hierarchy, names the design plans that are used by it and the constraints that its parameters may need to satisfy.

default selector will be used which selects plans in declaration order. Note that in this declaration, as with others, the order of the individual parts of the declaration may vary according to the user's wishes.

Plans

Each specialist has a collection of plans that may be selected depending on the situation, and it will follow the plan in order to achieve that part of the design for which it is responsible. A Plan consists of a sequence of calls to Specialists or Tasks (see below), possibly with interspersed constraints. It represents one method for designing the section of the component represented by the specialist. The specialists below will refine the design independently, tasks produce further values themselves, constraints will check on the integrity of the decisions made, while the whole plan gives the specific sequence in which the agents may be invoked. Typically as one goes down in the hierarchy, the plans tend to become fewer in number and more straightforward. An example of this is shown Figure 11.

```

(PLAN
  (NAME HEADDP1)
  (TYPE Design)
  (USED-BY Head)
  (SPONSOR HeadDP1S;
  (BODY HeadTubeSeat
    MountingHoles
    Bearings
    SealAndWiper
    AirCavity
    AirInlet
    (CHECK-CONSTRAINT Air)
    TieRodHoles
  (REPORT-ON Head)

```

Figure 11: Plan "HeadDP1". The design plan specifies the specialist that uses it, what criteria should be met for it to be chosen, and the plan steps.

The type of a PLAN can be "Design" or "RoughDesign" depending on which phase of the design the knowledge applies to. The SPONSOR's job is to give an opinion to a selector about how suitable this

plan is given the current state of the design. The BODY contains the details of the plan, and consists of an ordered list of plan items. In this example the plan consists entirely of tasks, with the exception of a constraint test and the last item which is a function provided by DSPL to print out the attributes and values of some part of the design.

Steps, Tasks, and Constraints

We consider a Step to be a design agent that can make one design decision given the current state of the design and taking into account any constraints. For example, one step would decide on the material for some subcomponent, while another would decide on its thickness.

```
(STEP
  (NAME      AirCavityID)
  (USED-BY   AirCavity)
  (ATTRIBUTE-NAME HeadAirCavityID)
  (REDESIGNER AirCavityIDRedesigner)
  (FAILURE-SUGGESTIONS
    (SUGGEST (DECREASE RodDiameter))
    (SUGGEST (DECREASE HeadBearingThickness))
    (SUGGEST (CHANGE HeadMaterial
                  TO DECREASE MinThickness))
  )
  (COMMENT "Find air cavity internal diam")

  (BODY
    (KNOWN
      BearingThickness
        (KB-FETCH 'Head 'HeadBearingThickness)
      RodDiameter
        (KB-FETCH 'Rod 'RodDiameter)
      HeadMaterial
        (KB-FETCH 'Head 'HeadMaterial)
      MinThickness
        (KB-FETCH HeadMaterial 'MinThickness)
    )
  )
  {Continued in next figure}
```

Figure 12: Step "AirCavityID".

A typical STEP in AIR-CYL is given in Figures 12 and 13. The STEP is broken down into two figures for convenience in display, and should be treated as one figure. A brief explanation of the role of the knowledge in the STEP is as follows. The sample STEP is USED-BY the AirCavity task. The ATTRIBUTE-NAME is the attribute for which this step is to design a value; that is, the Internal Diameter of the Air Cavity in the Head of the Air Cylinder. If a failure occurs the REDESIGNER will attempt to recover from it by altering the value just selected for this step's attribute. The declaration REDESIGN NOT-POSSIBLE is also allowed. If the step itself fails the FAILURE-SUGGESTIONS get passed up to the controlling task in a failure message. The suggestions refer to attributes that might be the cause of the failure. Each item in the suggestion list is evaluated at failure time. This allows conditional suggestions such as (IF (. x y) THEN (SUGGEST ...)). However, if the suggestion includes an expression, as in (DECREASE xyz BY (+ pqr 0.56)), then the SUGGEST function will arrange for the value to be computed. The actions DECREASE, INCREASE or CHANGE refer to attributes, such as RodDiameter. In the current system all attribute names must be unique.

The BODY of the step is divided in KNOWN and DECISION sections. The keywords KNOWN and DECISIONS will work just as well, and, in general, singular or plural keywords may be used as

```

(Continued from previous figure)
(DECISIONS
MaxRodRadius (VALUE+ (HALF RodDiameter))
MaxBearingThickness
(VALUE+ BearingThickness)
AirCavityRadius
(+ MinThickness
(+ MaxRodRadius
MaxBearingThickness))
AirCavityID (DOUBLE AirCavityRadius)
REPLY (TEST-CONSTRAINT ACID)
REPLY (KB-STORE
'Head 'HeadAirCavityID AirCavityID)
)
))

```

Figure 13: Step "AirCavityID" continued.

required. The KNOWN section obtains the values from the design data-base by doing KB-FETCH. The KB-FETCH uses the component and attribute names. The single quote (i.e., ') is used to indicate that the name given is to be used directly without evaluation, as opposed to the use of a variable (e.g., HeadMaterial) that should be evaluated prior to use (e.g., HeadMaterial) that should be evaluated prior to use (e.g., giving the value "Aluminum").

The DECISIONS section contains the design knowledge. It consists of variable-action pairs, where the action is evaluated and its value assigned to the variable. That variable may then be used in subsequent actions in the step. The variables set in the KNOWN section may also be used. Arithmetic expressions use prefix operators. The function VALUE+ returns the value plus the positive tolerance of the value, and consequently provides the largest magnitude for that value. There are many other functions available.

There are two distinguished variable names. One is REPLY, the other COMMENT. A comment will act as a dummy assignment and will expect a string as the action. This is just a way of inserting a comment into the body of the step. A REPLY variable is used when there is no value produced by the action but a message showing success or failure is produced instead. The TEST-CONSTRAINT and the KB-STORE are two examples. The value calculated by the step is put into the design data-base with a KB-STORE. It can produce a failure message if a constraint in the design data-base fails. Any failure will stop the execution of the body and will cause the DECISION section to fail.

A task is a design agent which is expressed as a sequence of steps, possibly with interspersed constraints. It is responsible for handling the design of one logically, structurally, or functionally coherent section of the component; for example a seat for a seal, or a hole for a bolt.

A Constraint is an agent that will test for a particular relationship between two or more attributes at some particular stage of the design. Constraints can occur at almost any place in the hierarchy. For example, a constraint might check that a hole for a bolt is not too small to be machinable given the material being used.

4.c.6 Other Features

The main purpose of this exposition is not to give a complete description of DSPL, but to give a feel for the task-specific nature of the tool. A few other essential features of the language will now be briefly described.

Failure Handling and Redesign capability is an important requirement for anything other than

relatively simple design problems. DSPL does not have failure analysis capabilities, but it can accept explicit knowledge about how to handle different kinds of failures during design. All design agents detect their own failure, are able to determine what went wrong (at least superficially), attempt to see if they can fix it locally, do so if they can, and report failure only if all attempts fail. Agents which have some control over other agents can use those agents in their attempt to correct the detected problem.

Each kind of agent can have different kinds of reasons for failing. For example, a step finds that a decision violates some constraint, a task discovers that a step's failure can't be mended locally, a plan can fail if it is discovered that it's not applicable to the situation to which it is being applied, while a specialist can fail if all of its plans fail.

For every kind of failure a message giving details is generated and passed back to the calling agent. The message includes, wherever possible, suggestions about what might be done to alleviate the problem. As there are usually many kinds of problems that can occur, an agent will first look at the message to decide what went on below. This is done by the Failure Handler associated with the agent. Much of the failure analysis is provided by the system, but for some cases, for example for constraint failures, the user (that is the person using the plan language to write a design system) has to supply some details. For some conditions immediate failure can be specified, for others an attempt to redesign might be made.

Knowledge about how to recover from failure can be coded as a *redesigner*. There appears to be a difference between the 'most reasonable choice' knowledge encoded in the step and the 'most reasonable adjustment' knowledge encoded in the redesigner. The language provides a number of constructs for representing failure handling knowledge of the above types.

Sponsors and selectors: A *sponsor* is associated with a plan and it is responsible for estimating the suitability of a plan for a particular design situation. A *selector* takes the output of the sponsor and will decide whether or not to use the plan if it has been recommended as suitable. The sponsor is expected to provide a suitability value for the plan, and if it cannot, a "use of plan language" failure will occur.

The *selector* takes as input the names of the plans being considered and their suitabilities for use in the design situation as decided by their sponsors. It will pick a plan for the specialist to execute.

4.c.7 Use and Extensions of DSPL

This section has discussed the idea of languages in which to express problem-solving knowledge, and has presented the language DSPL for a class of design problem-solving. DSPL embodies an underlying theory of routine design problem-solving. Examples of Specialist, Plan, Task, Step, Constraint, Redesigner, Failure Handler, Sponsor, and Selector knowledge were given. Most of these examples were taken from AIR-CYL, an expert system to design an Air Cylinder.

DSPL has been used for the construction of MPA, a system for routine logistics planning [11], and in the construction of a design system in the domain of chemical engineering [32].

More work needs to be done to test the applicability of this language to other design problems and other domains. Extensions to the theory must be made in order to handle design activity which is not of the type where both problem-solving and knowledge are known in advance. We feel that the identification of some of the types of design knowledge and their use is a substantial contribution towards understanding routine design activity.

5 The Generic Task Toolset

5.a Important Properties of the Toolset

So far, we have outlined the generic task theory and also described two such generic task tools: CSRL and DSPL. Tools corresponding to other generic tasks are also in existence in varying degrees of completeness. These tools are currently available for the Interlisp/Loops environment in the Xerox 1100 series of Lisp machines. CSRL and DSPL are also available in versions that are compatible with the KEE development system of Intellicorp. CSRL is also available in Commonlisp⁷. Currently, a project is under way at our Laboratory for the entire toolset to be made available in Commonlisp.

The integrated generic task toolset is extensible in the sense that more generic tools can be added as they are invented and additional problem solvers can be invoked as needed. The tools are intended to ensure the following advantages of the generic tasks, as described in [8].

- *Multiformity.* The more traditional architectures for the construction of knowledge based systems emphasize the advantages of uniformity of representation and inference. However, in spite of the advantage of simplicity, we argued earlier that uniformity results in a level of abstraction problem. A uniform representation cannot capture important distinctions between different kinds of problems. A uniform inference engine does not provide different control structures for different kinds of problems.

The generic task approach provides multiformity. Each generic task provides a different way to organize and use knowledge. The knowledge engineer can choose which generic task is the best for performing a particular function, or can use different generic tasks for performing the same function. Different problems can use different generic tasks and different combinations of generic tasks.

- *Modularity.* A knowledge-based system can be designed by making a functional decomposition of its intended problem solving into several cooperating generic tasks, as illustrated in our discussion on diagnosis. Each generic task provides a way to decompose a particular function into its conceptual parts, e.g., the categories for hierarchical classification, and allows domain knowledge of other forms to be inserted into a generic task, e.g., evidence combination knowledge in hierarchical classification [39]. Each generic task localizes the knowledge that is used to satisfy local goals.
- *Knowledge Acquisition.* Each generic task is associated with its own knowledge acquisition strategy for building an efficient problem solver [7]. For example in hierarchical classification, the knowledge engineer needs to find out what specific categories should be contained in the

⁷CSRL is available as a supported product from Battelle Memorial Laboratories, Artificial Intelligence Group, in Columbus, Ohio, including versions in Commonlisp.

classification hierarchy and what general categories provide the most leverage for the establish-refine strategy.

- *Explanation.* This approach directly helps in providing explanations of problem solving in expert systems in two important ways: how the data match local goals and how the control strategy operates [15]. Also, the control strategy of each generic task is specific enough for generating explanations of why the problem solver chose to evaluate or not to evaluate a piece of knowledge. This is because of the higher level of abstraction in which control is specified for generic tasks.
- *Exploiting Interaction between Knowledge and Inference.* Rather than trying to separate knowledge from its use, each generic task specifically integrates a particular way of representing knowledge with a particular way of using knowledge. This allows the attention of the knowledge engineer to be focused on representing and organizing knowledge for performing problem solving.
- *Tractability.* Under reasonable assumptions, each generic task generally provides tractable problem solving [1, 21]. (One major exception is abductive assembly, which can become intractable under certain conditions, making it hard then for humans and machines to perform the task.) The main reasons why they are tractable are that a problem can be decomposed into small, efficient units, and knowledge can be organized to take care of combinatorial interactions in advance.

It should be noted that these advantages are attained at the cost of generality. Each generic task is purposely constrained to perform a limited type of problem solving and requires the availability of appropriate domain knowledge.

5.b Integrating and Combining GT's in an Application

It is hard at this overview level to give enough details of how the tools are to be combined to put together complex applications. There are both significant theoretical issues of integration as well as practical issues of technology and implementation in this regard. The diagnosis example is a good example to discuss the practical issue of how the tools in the toolset can be combined for an application.

We need to make the following distinctions that will be helpful here. Each tool such as CSRL can be regarded as a "shell" of a particular p.s. type. When the tool is invoked and knowledge and inference encoded using the knowledge primitives and message types, we have a *problem solver*, and different problem solvers built with the same shell can (and will typically) exist in an application. Each of the problem solvers is a specialist in two different senses: it specializes in a particular body of knowledge and in a type of problem solving, e.g., the *automech* (domain/concept) hierarchical classifier (type of problem solving) built out of CSRL, or the *BadFuel* (domain/concept) hypothesis matcher (type of problem solving) built out of HYPER, or the *AIRCYL* (domain/concept) hierarchical designer (type of problem solving). A given problem solver will typically need to access other problem solvers for information to

continue its problem solving, e.g., the *BadFuel* matcher will need to know if various specific data items are present and for this it will need to access the data inference problem solver built from the tool IDABLE.

When the KBS is being built using the tools in the toolset, the knowledge engineer controls and directs the interaction among problem solvers by shaping and directing the messages appropriately. Without getting into details, the idea is simple: Each problem solver is characterized by (i) the kind of questions it can accept: e.g., the *Badfuel* matcher can accept messages that concern confidence values about the *Badfuel* concept and (ii) the kind of questions that it can ask, e.g., *Badfuel* concept can ask the database problem solver for values of specific data attributes. In the current version of the toolset these decisions have to be explicitly made by the knowledge engineer, i.e., which problem solver has to send what types of queries to what other problem solvers has to be specified at the time the system is being built. The toolset itself is built on top of a substratum that is object- and message-oriented so that building additional generic tools within the framework is a straightforward thing to do. See [26] for details on how the toolset is built up in this way.

For a complex application which involves portions which match the problem solving behaviors of the tools in the toolset, but which also has portions which require other methods of reasoning and representation not included within the current toolset, escaping to the object level or even the Lisp level (as in current implementations) to program AI or numerical techniques may be necessary and the toolset implementation supports this.

6 Concluding Remarks

In the late 70's, when we embarked on this line of research -- characterized by an attempt to identify generic tasks and the forms knowledge and control required to perform them -- the dominant paradigms in knowledge-based systems were rule and frame type architectures. While our work on use-specific architectures was evolving, dissatisfaction at the limited vocabulary of tasks that these architectures were offering was growing at other research centers. Clancey [17] in particular noted the need for specifying the information processing involved by using a vocabulary of higher level tasks. Task-level architectures have been gathering momentum lately: McDermott and his coworkers [28] have built SALT, a shell for a class of design problems, where critiquing proposed designs by checking for constraint-violations is applicable. Clancey [18] has proposed a shell called Heracles which incorporates the heuristic classification strategy for diagnosis. Bennett [2] presents COAST, a shell for the design of configuration problem solving systems. All these approaches share the basic thesis of our own work, viz., the need for task-specific analyses and architecture support for the task. However, there are some differences in assumptions and methodology in some cases that needs further discussion.

The following conceptual distinctions are useful:

- "Building blocks" out of which more complex problem solvers can be composed, such as the tasks in the theory presented in the paper.
- Explicit high level strategies which we want a system to follow, where the strategies are expressed in terms of some set of tasks. Clancey's Heuristic Classification is an example of this. McDermott's and Marcus' Salt system uses a strategy called "propose and refine" which is also of this type.
- Compound tasks, such as the form of diagnosis described in earlier in the paper. An architecture for this compound task will bring with it its constituent generic tasks and also help in integrating the problem solving from the viewpoint of the overall task. In our

Laboratory, we are at work on building such diagnostic-level problem solving architectures for diagnosis of process engineering systems.

- Tasks which do not necessarily correspond to those human experts do well, but nevertheless can be captured as appropriate combinations of knowledge and inference and a clear function can be associated with them, e.g., constraint satisfaction schemes. Bennet's Coast system is an example of this.

Once we identify task-level architectures as the issue for highest leverage, then a number of immediate questions arise: what is the criterion by which a task is deemed to be not only generic but is appropriate for modularization as an architecture? How about an architecture for the generic task of "investment decisions"? Diagnosis? Diagnosis of process control systems? Is uncertainty management a task for which it will be useful to have an architecture? Are we going to proliferate a chaos of architectures without any real hope of reuse? What are the possible relationship between these architectures? Which of these architectures can be built out of other architectures? I do not propose to answer all these questions here, but they seem to be the appropriate kinds of questions to ask when one moves away from the comfort of universal architectures and begins to work with different architectures for different problems.

At this stage in the development of these ideas, empirical investigation of different proposals from the viewpoint of usefulness, tractability and composability is the best strategy. From a practical viewpoint, any architecture that has a useful function and for which one can identify knowledge primitives and an inference method ought to be considered a valid candidate for experimentation. As the tools evolve, one may find that some of the architectures are further decomposable into equally useful, but more primitive, architectures; or that some of them do not represent particularly useful functionalities, and so on.

The generic tasks that are represented in our toolbox were specifically chosen to be useful as technology for building diagnosis, planning and design systems with compiled expertise. For capturing intelligent problem solving in general, we will undoubtedly require many more such elementary strategies and ways of integrating them. For example, the problem solving activities in qualitative reasoning and device understanding, e.g., qualitative simulation, consolidation, and functional representation. All these tasks have well-defined information processing functions, specific knowledge representation primitives and inference methods. Thus candidates for generic information processing modules in our sense are indeed many.

What does all this mean for an architecture of intelligence?

I am led to a view of intelligence as an interacting collection of *functional units*, each of which solves an information processing problem by using knowledge in a certain form and corresponding inference methods that are appropriate. Each of these units defines an information processing faculty. I discuss elsewhere [14] the view that these functional units share a computational property: they provide the agent with the means of transforming essentially intractable problems into versions which can be solved efficiently by using knowledge and inference in certain forms. For example, Goel et al [21] show how classification problem solving solves applicable cases of diagnosis with low complexity, while diagnosis in general is of high complexity. Knowledge is indeed power, but how it acquires its power is a far subtler story than the first generation knowledge based systems made it appear.

This view generates its own research agenda: As a theory, the generic tasks idea has quite a bit of work ahead of it in terms of a coherent story about how the tasks come together, are integrated and how more complex tasks such as planning come about from more elementary ones. How complex inference methods develop from simpler ones and how learning shapes these functional modules are issues to be investigated.

The relationship of task-specific architectures such as the GT ideas in this paper to, on one hand, more general architectures, such as SOAR [33], and on the other to "weak methods" is an intriguing one. My view is that from the perspective of modeling cognitive behavior, a GT-level analysis provides two closely related ideas which give additional content to phenomena at the SOAR architecture level. On the one hand, the GT theory provides a vocabulary of goals that a SOAR-like system may have. On the other hand, this vocabulary of goals also provides a means of indexing and organizing knowledge in Long Term Memory such that when SOAR is pursuing a problem solving goal, appropriate chunks of knowledge and control behavior are placed in Short Term Memory for SOAR to behave like a GT problem solver. In this sense a SOAR-like architecture, based as it is on goal achievement and universal subgoalting, provides an attractive substratum on which to implement future GT systems. In turn, the SOAR-level architecture can give graceful behavior under conditions that do not match the highly compiled nature of GT-type problem solving.

Acknowledgements

I am indebted to my colleagues with whom I have written a number of papers over the years on the generic task approach. In the preparation of this paper I have used excerpts from such papers. In particular the papers by the author and W. Punch, the author and D. C. Brown, the author, T. C. Bylander and John Josephson have been used. I acknowledge the assistance of Richard Fox and Vibhu Mittal in the preparation of this version of the paper. Matt DeJongh helped with some sections of this paper. The comments of John Fox, the editor of *Knowledge Engineering Review*, have helped improve the paper substantially. I also gratefully acknowledge the support of the Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010, and the Air Force Office of Scientific Research, grant 87-0090.

References

- [1] Dean Allemang, Michael C. Tanner, Tom Bylander, and John R. Josephson.
On the Computational Complexity of Hypothesis Assembly.
January, 1987
Proceedings of IJCAI-87, Milan, Italy, August, 1987.
- [2] James Bennett.
COAST: A Task-Specific Tool for Reasoning About Configurations.
In *Proc. AAAI Workshop on High-Level tools*. AAAI, Shawnee Park, Ohio, 1986.
- [3] Brown, D. C.
Expert Systems for Design Problem-Solving using Design Refinement with Plan Selection and Redesign.
PhD thesis, The Ohio State University, August, 1984.
- [4] Brown, David C. and Chandrasekaran, B.
Knowledge and Control for a Mechanical Design Expert System.
IEEE Computer 19:92-101, July, 1986.
- [5] Buchanan, B., Sutherland, G. and Feigenbaum, E.A.
Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses.
Organic Chemistry, 1969.
- [6] Bylander, T., and S. Mittal.
CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling.
AI Magazine 7(3):66-77, 1986.
- [7] Bylander, T., Smith J. and Svrbely, J.
Qualitative Representation of Behavior in the Medical Domain.
In *Proceedings of The Fifth Conference on Medical Informatics*, pages 7-11. Conference on Medical Informatics, Washington, D.C., October 26-30, 1986.
- [8] Tom Bylander and Todd R. Johnson.
Structured Matching.
1987
OSU CIS LAIR Technical Report.
- [9] Chandrasekaran, B.
Towards a Taxonomy of Problem Solving Types.
AI Magazine :9-17, Winter/Spring, 1983.
- [10] Chandrasekaran, B.
Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design.
IEEE Expert 1(3):23-30, 1986.

- [11] Chandrasekaran, B., and Tanner, M.
Uncertainty Handling in Expert Systems: Uniform vs. Task-Specific Formalisms.
Uncertainty in Artificial Intelligence.
North Holland Publishing Company, 1986, pages 35-46.
Kanal, L.N. and Lemmer, J. (Editors).
- [12] B. Chandrasekaran.
Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1183-1192. Milan, Italy, August, 1987.
- [13] Chandrasekaran, B., Smith, J., Sticklen, J.
Deep Models and Their Relation to Diagnosis.
Technical Report, , 1987.
Invited paper, Toyota Foundation Symposium on Artificial Intelligence in Medicine, Tokyo, Japan, August 1986, available as technical report from Laboratory of AI Research, Ohio State University.
- [14] Chandrasekaran, B.
What Kind of Information Processing is Intelligence? A Perspective On AI Paradigms and A Proposal.
This paper will appear in Source Book on the Foundations of AI, Partridge and Wilks, Editors, Cambridge University Press.
1987
- [15] Chandrasekaran, B., M. C. Tanner and J. R. Josephson.
Explanation: the Role of Control Strategies and Deep Models.
Expert Systems: The User Interface.
Ablex Publishing Corporation, Norwood, New Jersey 07648, 1988, pages 219-247.
Editor, James Hendler.
- [16] Chandrasekaran, B.
Design: An Information Processing Level Analysis.
Technical Report, The Ohio State University, Columbus, Ohio 43210, 1988.
- [17] Clancey, W.J.
NEOMYCIN: Reconfiguring a Rule-Based Expert System for Application to Teaching.
In *Proc. Seventh International Joint Conference on Artificial Intelligence*, pages 829-836. IJCAI, Vancouver, 1981.
- [18] Clancey, W. J.
Heuristic Classification.
Artificial Intelligence 27(3):289-350, 1985.
- [19] Duda, R.O., Gaschnig, J.G., and Hart, P.E.
Model Design in the Prospector Consultant System for Mineral Exploration.
Expert System in the Microelectronic Age.
Edinburgh University Press, 1980, pages 153-167.
Editor, Michie, D.
- [20] Friedland, P.
Knowledge-based Experiment Design in Molecular Genetics.
PhD thesis, Stanford University, Computer Science Department, 1979.
- [21] Goel, A., N. Soundararajan, and B. Chandrasekaran.
Complexity in Classificatory Reasoning.
In *Proc. National Conference on Artificial Intelligence*, pages 421-425. Seattle, Washington, July 13-18, 1987.

- [22] Gomez, F. and B. Chandrasekaran.
Knowledge Organization and Distribution for Medical Diagnosis.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1):34-42, January, 1981.
- [23] Hashemi, S., Hajek, B.K., Miller, D.W., Chandrasekaran, B., and Josephson, J.R.
Expert Systems Application to Plant Diagnosis and Sensor Data Validation.
In *Proceedings of the Sixth Power Plant Dynamics Control and Testing Symposium*. Knoxville, Tennessee, April, 1986.
- [24] Johnson, T.
HYPER: The Hypothesis Matcher Tool.
In *Proceedings of Expert Systems Workshop*, pages 122-126. Defense Advanced Research Projects Agency, Pacific Grove, CA, April 16-18, 1986.
- [25] Johnson, K., Stickien, J., and Smith, J.W.
IDABLE -- Application of an Intelligent Data Base to Medical Systems.
In *Proceedings of the AAAI Spring Artificial Intelligence in Medicine Symposium*, pages 43-44. American Association for Artificial Intelligence, Stanford University, March 22-24, 1988.
- [26] Josephson, J.R., Smetters, D., Welch, A.K., Fox, R., Flores, G, and Lyndes, D.
Generic Task Toolset DRACO Release - Beta Test Including RA.
Technical Report, The Ohio State University, Computer & Information Science Department, Laboratory for Artificial Intelligence Research, February, 1988.
- [27] Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C.
A Mechanism for Forming Composite Explanatory Hypotheses.
IEEE Trans. on Systems, Man and Cybernetics :pp.445-454, 1987.
- [28] Marcus, Sandra, and John McDermott.
SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems.
Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1987.
- [29] J. McDermott.
R1: A Rule-based Configurer of Computer Systems.
Artificial Intelligence 19(1):39-88, 1982.
- [30] Miller, R.A., Pople, H.E., and Meyers, J.D.
Internist-1, An Experimental Computer-based Diagnostic Consultant for General Internal Medicine.
Readings in Medical Artificial Intelligence.
Addison-Wesley Publishing, 1984, pages 190-209.
- [31] Mittal, S.
Design of A Distributed Medical Diagnosis and Data Base System.
PhD thesis, The Ohio State University, 1980.
- [32] Myers, D.R., Davis, J.F., and Herman, D.
A Task Oriented Approach to Knowledge-Based Systems for Process Engineering Design.
Computers and Chemical Engineering, Special Issue on AI in Chemical Engineering Research and Development : , August, 1988.
- [33] Laird, J.E., Newell, A., Rosenbloom, P.S.
SOAR: An Architecture for General Intelligence.
Artificial Intelligence 33:1-64, 1987.
- [34] Punch, W. F., Tanner, M. C., Josephson, J. J.
Design Considerations for PEIRCE, A High-Level Language for Hypothesis Assembly.
Expert Systems in Government Symposium :279-281, October, 1986.

- [35] Shortliffe, E.H.
Computer-based Medical Consultations: MYCIN.
Elsevier/North-Holland Inc., 1976.
- [36] Shum, S.K., Davis, J.F., Punch III, W.F., and Chandrasekaran, B.
An Expert System Approach for Malfunction Diagnosis in Chemical Plants.
Computers and Chemical Engineering 12(1):27-36, 1988.
- [37] J. W. Smith, M.D., J. R. Svrbely, C. A. Evans, P. Straum, J. R. Josephson and M. C. Tanner.
RED: A Red-Cell Antibody Identification Expert Module.
Journal of Medical Systems 9(3):121-138, 1985.
- [38] Sticklen, J., Chandrasekaran, B. and Josephson, J.
Control Issues in Classificatory Diagnosis.
In *Proceedings of The 9th International Joint Conference on Artificial Intelligence*. International
Joint Conference on Artificial Intelligence, University of California, Los Angeles, CA, August
18-24, 1985.
- [39] Jon Sticklen.
MDX2: An Integrated Medical Diagnostic System.
June, 1987
Phd. Dissertation, Department of Computer and Information Science, The Ohio State University,
Columbus, Ohio 43210.
- [40] Tanner, M. and Bylander, T.
Application of the CSRL Language to the Design of Expert Diagnosis Systems: The Auto-Mech
Experience.
Artificial Intelligence in Maintenance.
Noyes Publications, Park Ridge, N.J., 1985.

Appendix E

Artificial Intelligence Perspectives on the Classification Task

From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task

B. CHANDRASEKARAN, FELLOW, IEEE, AND ASHOK GOEL

Abstract—We consider the very general information processing task of classification, and review it from the perspectives of the knowledge-based reasoning, pattern recognition, and connectionist paradigms in artificial intelligence, paying special attention to knowledge-based classificatory problem solving. We trace the evolution of the mechanisms for classification as the computational complexity of the problem increases, from numerical parameter setting schemes, through those using intermediate abstractions and then relations between symbols, and finally to complex symbolic structures that explicitly incorporate domain knowledge. The paper can be viewed as a bridge-building activity, describing the approaches of three different research communities to the same general task.

I. INTRODUCTION

CLASSIFICATION is a very general information processing task in which specific entities are mapped onto general categories. As the amount of data about the entity to be classified and the number of classificatory categories increase, typically so does the computational complexity of the task. In this paper, we review the classification task from the perspectives of the knowledge-based reasoning, pattern recognition, and connectionist paradigms in artificial intelligence (AI), paying special attention to knowledge-based classificatory problem solving. We trace the evolution of the mechanisms for classification as the complexity of the problem increases, from numerical parameter setting schemes, through those using intermediate abstractions and then relations between symbols, and finally to complex symbolic structures that explicitly incorporate domain knowledge. The paper can be viewed as a bridge-building activity, describing the approaches of three different research communities to the same general task. It can also be viewed as an attempt, by using the classification task as a concrete example, to give an intuitive account of how the information processing activity underlying thought necessarily evolved into complex symbolic processes in order to handle increasing complexity of problems and requirements of flexibility.

Manuscript received April 20, 1986; revised January 30, 1988. This work supported in part by the Defense Advanced Project Research Agency contract RADC F30602-85-C0010, and in part by the Air Force Office of Scientific Research, grant 87-0090.

The authors are with the Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, Ohio State University, Columbus, OH 43210-1277.

IEEE Log Number 8820776.

II. THE CLASSIFICATION TASK

Classification, sometimes called categorization in the cognitive science literature, as an information processing task can be functionally specified by the information it takes as input, and the information it gives as output. In its general form, the input to the classification task is a collection of data about some specific entity (e.g., an object, a state, a case, or a situation), and the output is the general category (or categories) pertaining to the entity. We note that this characterization of the classification task as a map from specific entities to general categories makes no commitments to the mechanism by which the mapping is to be accomplished. Classification has been an active research issue in the knowledge-based reasoning, pattern recognition, and connectionist paradigms, though the paradigms differ in the mechanisms by which the task is performed.

A. Classification and Knowledge-Based Systems

The area of knowledge-based reasoning, though of relatively recent origin, is already a well established paradigm in AI. The essential idea of the field is to capture in computer programs, explicitly and in symbolic form, the knowledge and *problem solving methods* of human experts for selected domains and tasks. In fact, because of the central role of explicit domain knowledge of human experts, the field is often called expert systems. This is not an appropriate place to discuss the general issues of knowledge representation and problem solving in the area of knowledge-based systems, many of which remain open and active research issues. There are many expert tasks that have been successfully emulated by these systems; there are an even larger number of things that human experts do that are beyond the current state of technology for construction of knowledge-based systems. Nevertheless, when we examine the intrinsic nature of the *tasks* that knowledge-based systems perform, a surprising fact emerges: many of them solve variants of problems which are intrinsically classificatory in nature. We are not suggesting here that the authors of these programs recognized them as classification problems and used methods appropriate to the classification task, but that independent of how they

were solved the problems have an intrinsically classificatory character. Let us consider some examples.

- The MYCIN system [36], in its diagnostic phase, has the task of classifying patient data onto an infectious agent hierarchy, i.e., the diagnostic task is identification of an infectious agent category, as specific as possible, that pertains to the patient data.
- The PROSPECTOR system [14] classifies a geological description as corresponding to one or more mineral formation classes.
- The SACON system [3] classifies structural analysis problems into categories for each of which a particular family of analytical methods is appropriate.
- The MDX system [6], [8], [20] explicitly views a significant portion of the diagnostic task as classifying a complex symbolic description (the patient data) as an element, as specific as possible, in a disease classification hierarchy.

We do not mean to imply that all problems are classification problems, or that they can be usefully converted into such problems. R1 [27] and AIR-CYL [5], e.g., perform different versions of the object synthesis problem, i.e., simple versions of the design problem. Dendral [4], Internist [30] and RED [22] are different systems all performing various versions of abductive assembly of composite explanatory hypotheses. Chandrasekaran [7], [9], [10] has provided taxonomies of such generic tasks, and has identified classification as one of them. Recently, Clancey [12] has made a similar assessment of how several knowledge-based systems perform classificatory problem solving.

B. Classification and Pattern Recognition Models

The area of pattern recognition, now nearly 30 years old, represents another paradigm in AI. The classification task has been intimately associated with pattern recognition models from the very beginning of the field. In fact, in the early days of AI, the problem of recognition was formulated as a problem of classification, in particular one of statistical classification of pattern vectors onto one of a finite number of categories, each category characterized by some kind of probability distribution. Indeed, what started out as a practically useful formulation became so dominant that there was a need for a paper such as that by Kanal and Chandrasekaran [23] pointing out that classification is only one of the formulations for the more general recognition problem. Even when newer techniques such as syntactic techniques came into the field, the problem was still often formulated as a classification problem, this time into grammatical categories.

C. Classification and Connectionist Networks

Neural modeling, which predates the early perceptron models and appears to be undergoing a revival in its modern connectionist version, is still another paradigm in AI. The essential idea in this area is to represent knowl-

edge as numerical weights of connections between units in a network. A variety of neural models, from linear threshold networks [15], [31], [32], to nonlinear analogue architectures [21], have been developed. These models typically deal with motor or perceptual phenomena; neural networks that capture a range of complex, higher-level cognitive processes have yet to be proposed. Although our remarks are intended to be more generally applicable, in this paper we will confine our discussion only to linear threshold, digital networks in the connectionist mold in which the emphasis is on the memory and learning aspects of reasoning.

The earlier connectionist networks, e.g., the perceptron model, were once viewed as devices for practical visual pattern recognition, and since the problem of pattern recognition itself was viewed as that of classification, perceptrons were really classificatory devices. The important role of classification is evident even in the more recent connectionist architectures, in which "hidden" units separate the input and the output units. Let us consider, as an example, the NETtalk system [34], a connectionist scheme for the task of word pronunciation. It uses a numerical relaxation technique for problem solving, and a method for back propagation of corrective feedback during learning. The important point for our purposes, however, is that NETtalk performs its task by *classifying* character substrings of the input words onto phonemes.

III. THE UBIQUITY OF CLASSIFICATION

There are two things that are important to note from the above discussion: firstly, classification appears to be a rather ubiquitous information processing task, and secondly, classification has been an important research issue in the various paradigms in AI. This suggests that classification is not an artifact of any one point of view, but rather a "natural kind" of information processing task of considerable cognitive significance. Indeed, classification appears to be a powerful human strategy for organizing knowledge for comprehension and action. The human tendency to classify input entities is so strong that we often classify without necessarily being consciously aware of it, and feel we have accomplished something by merely naming entities as categories, even if we cannot do much about it. The use of classification as a strategy for knowledge organization can be found in virtually every area of human intellectual activity. In biology, e.g., taxonomic classification has long been an important methodology for organization of knowledge, and recently, mathematical techniques has been pressed into service for providing better classification in this field [37]. Some of the more recent controversies regarding evolutionary biology, e.g., the traditional gradual evolutionary vs. the punctuated equilibrium theories, also revolve around implications of various theories of biological classification. The periodic table of chemical elements is another common classification structure in which first groups of elements and then the specific elements are identified.

A. The Computational Power of Classification

A simple computational explanation can be given for the importance of classification as an information processing strategy. We can think of a general task of an intelligent agent as performing actions on the world for achieving certain goals, where the right action for accomplishing a specific goal typically is a function of the relevant states of the world. In the medical domain, for example, we may view the general problem facing the physician as that of finding an appropriate therapeutic action for a given set of symptoms that describes the state of a patient and is a subset of the set of all possible symptoms. One way of mapping states of the world to actions on it might be to use a *decision table* that relates various subsets of state variables to the action variable. However, if there are n state variables s_1, s_2, \dots, s_n , each of which may take on one of q values, then both the time and space complexities of mapping the states onto actions by table look-up are $O(n \cdot q^n)$ [17]. Thus, the table look-up approach to making decisions about actions on the world would be useful only for very small problems. In fact, the cardinality of the relevant states of the world generally is very large, e.g., in the medical domain, the total number of possible states of a patient is the Cartesian product of the distinct values for each of the state variables (symptoms, values from laboratory tests, other manifestations etc.). Thus, for complex, real world problems such as medical problem solving the decision table is bound to be too large for construction, storage, looking up, and modification.

The general problem of finding the right action may be solved more efficiently, however, if action knowledge can be indexed, not by the states of the world, but by equivalence classes of states of the world. A physician's therapeutic knowledge, e.g., may be indexed not directly by the detailed values of the patient state variables, but by diseases, each of which can be thought of as defining an equivalence class of patient state variables. We are proposing a functional decomposition into two stages, of the overall task of mapping from states of the world to actions on it: first, mapping the states onto their equivalence classes for indexing the right actions. This decomposition often results in substantial reductions in the computational complexity of the problem, since the number of equivalence classes is typically much smaller than the number of states. The classification task corresponds to the first component in this decomposition, in which specific entities such as states of the world are mapped onto general categories that represent their equivalence classes. Medical problem solving thus may be organized first as classifying patient symptoms onto disease categories, i.e., diagnosis as classification, and then indexing the therapeutic actions by the disease categories. It may not, of course, always be possible to decompose the general problem of finding the right action in such a manner; however, whenever possible, it is computationally advantageous to do so. The decomposition of mapping states of the world to actions on it is illustrated by the JESSE system [18], which supports a

simple version of political decisionmaking. JESSE first classifies the state variables describing a given situation onto situation assessment categories, and then uses these categories to index appropriate policies for action from a store of policy options.

B. Classificatory Categories

Classificatory categories represent the equivalence classes of entities that are input to the classification task. Much of human thinking is organized around classification, both in terms of acquiring new classificatory categories, and using existing categories to perform classifications, since classification provides a substantial computational advantage in solving problems. In knowledge-based systems, the classificatory categories typically are labeled symbolically, and often correspond to *concepts* in the task domain. In connectionist networks on the other hand, no labels are associated with the categories, and the categories do not necessarily correspond directly with the domain concepts. The process of creating useful classificatory categories by concept learning is generally a much harder process than using an existing classification structure. Thus, in medicine, discovery of a disease, i.e., creation of a new category, is a relatively major event while diagnosis is much more routine. How these classificatory categories are created is an issue in research on learning and deep cognitive models [35]. In this paper we will deal only with the process of assigning an entity to an existing category in a classification structure.

IV. NUMERICAL APPROACHES TO CLASSIFICATION

So far we have discussed what is classification and why it is useful, but not how classification is accomplished, i.e., we have presented the forms of input and output information for the classification task, and have provided an explanation for the usefulness of classification as a strategy, but have not presented any mechanism for performing the task. In the remainder of this paper we will review various knowledge-based, pattern recognition, and connectionist approaches to classification. In this section we will discuss numerical parameter setting approaches to classification. In the next section we will show how the use of intermediate abstractions reduces the computational complexity of performing the classification task, and discuss why symbols may be used to capture these abstractions. In Section VI we will discuss the use of syntactic and structural relations between symbols for classification, and in Section VII we will provide a detailed account of how complex symbolic structures that explicitly incorporate domain knowledge may be used for classification.

A. Statistical Pattern Recognition

Most early pattern recognition models used the statistical approach to classification [13] in which the object of unknown classification is represented as a multidimensional pattern vector. Each dimension of the vector repre-

sents an attribute of the entity, and typically is represented as a numerical variable, even though ordinals are sometimes used. The choice of the attributes of the entity is such that they have the potential to distinguish between the categories, where each category is characterized by some kind of probability distribution. In the task domain of medical diagnosis, e.g., if it is desired to distinguish between diseases D_1 and D_2 , and the system designer has reason to believe that symptoms s_1, s_2, \dots, s_n carry useful information for this discrimination, then often careful statistical data gathering is possible such that a discriminant function of the variables s_1, s_2, \dots, s_n is a very accurate classifier. When the number of dimensions is small, it is possible to design statistical classification systems that outperform human performance, since human reasoning with the same number of variables may be less efficient in information extraction. Despite the enormous intrinsic interest in the mathematical problem of designing classification algorithms in the discriminant function framework, Kanal and Chandrasekaran [24] have pointed out that the real computational power often comes from a careful choice of the attributes based on a good knowledge of the domain, rather than from the specific design of the separation algorithm.

What happens when the dimensionality of the pattern vector becomes very large, or the number of categories becomes large? When the number of categories increases, then in order to make more and more distinctions, generally the number of measurements on the entity of interest, i.e., the dimensionality of the pattern vector, also needs to grow rapidly. The computational complexity of the algorithm to make the discrimination grows even more rapidly than the increasing number of dimensions, and correspondingly, the average performance, i.e., the correct classification rate, deteriorates quite rapidly. Sensitivity problems become quite severe, i.e., the required precision of the variables in the classification algorithm becomes impractically high. Opacity problems result, i.e., it becomes increasingly hard to make any kind of statement about what attributes are playing what role in the recognition process. Szolovits and Pauker [40], discuss these and some of the other problems with probabilistic approaches to classification.

B. The Perceptron Model

Roughly in parallel with the development of statistical approaches to classification in the pattern recognition paradigm came the development of the early connectionist models of classification, specifically, the perceptron model. The perceptron architecture [31], consists of a set of input units and an output unit, each unit being a two-state, linear threshold digital device. Each unit in the input layer is connected directly to the output unit, with some numerical weight associated with each such connection. The inputs to the perceptron are points in an orthographic projection of the object to be classified, where each input unit scans some points in the projection. The output is the truth

value of some predicate such as the predicate stating that the object, a_x , of unknown classification belongs to some known category, C_y . The numerical weights associated with the connections in the network act as parameters of the network, and collectively represent the discriminant function for classification of the input object onto different categories. The output of the network is computed by a linear combination of the evidence that flows into the output unit via the connections. The perceptron architecture can be trained to "learn" the discriminant function by appropriately adjusting the weights of the connections in the network. Feedback on whether the network has reached the correct classificatory conclusion is provided by the trainer during the learning sessions. It has been shown that if the input objects are linearly separable then the weights of the connections will converge to the discriminant function that can correctly distinguish between the objects in finite time.

When the number of categories and the number of points scanned on the objects to be classified are small then the perceptron can be powerful classifier, at least for linearly separable objects. However, when these numbers get larger then the perceptron suffers from problems similar to those in the statistical approaches to classification. As the number of categories increase, the number of points needed to be scanned by the input units for learning the discriminant function increases, which results in a rapid increase in the number of input units. The time complexity of learning the right weights for correct classification grows even more rapidly, and correspondingly, the correct classification rate drops rapidly for a fixed number of input units. The sensitivity problem worsens, i.e., even slight errors in the weights of the connections may result in large changes in the output. The opacity problem, i.e., recognizing specifically which weight is playing precisely what role in the classification process, hard in the perceptron model in any case, becomes even harder. Minsky and Papert [28] discuss the computational properties of the perceptron architecture, and point out some of the problems with it.

V. USE OF INTERMEDIATE ABSTRACTIONS IN CLASSIFICATION

The above discussion shows that while numerical parameter setting schemes may lead to powerful classifiers for small problems, the complexity of the separation algorithm becomes impractically high as the number of classificatory categories increases. The problem here is not so much in the specific choice of one discriminant function over another, but in the fact that these approaches seek to directly map the input entity onto classificatory categories. Indeed, similar complexity problems arise for all approaches that perform classification by directly mapping specific entities onto general categories. Let us consider, as another example of such direct classification, the method of discrimination tree traversal for medical diagnosis. Again, let the input be characterized by n state variables, s_1, s_2, \dots, s_n each of which can take on one of q values.

The state variables are organized in a tree in which the top node corresponds to some state variable s_1 and has q branches coming out of it, one for each of the q possible values that s_1 may take. The branches lead to q different nodes, each of which corresponds to some s_2 and has q branches coming out of it. This organization is repeated until all the state variables have been represented on the tree. Each of the q^n branches coming out of the q^{n-1} nodes at the n th level leads to one of a finite number of disease categories, D_1, D_2, \dots, D_m . The time and space complexities for classification by discrimination tree traversal are given by $O(n)$ and $O(q^n)$, respectively [17]. Clearly, for complex, real world problems, where the number of classificatory categories typically is large, the proposition of directly mapping input entities onto classificatory categories is quite futile.

What, then, can be done when the number of classificatory categories is large? Let us consider, as an example, the problem of automatic reading of texts in some language that consists of a large number of words. Intuitively, one would think that first recognizing characters (or perhaps substrings of characters) in the words, and then recognizing word themselves would be computationally more attractive. The words (or perhaps word-phrases) may be later used in understanding complete sentences in the language. In this approach, instead of performing classification by a direct mapping from the input entity onto the categories, intermediate abstractions are first constructed, the entity of unknown classification mapped onto these abstractions, which are then used as inputs to a higher-level classification process. What we are suggesting here is a conceptual decomposition of the classification process onto hierarchically organized intermediate abstractions. Such a conceptual decomposition makes the classification process more efficient, as we will see a little later.

A. Signature Tables

In order to make the notion of conceptual decomposition of the classification process into hierarchically organized intermediate abstractions more explicit, let us consider evaluation functions in game playing, e.g., playing chess, as another example of classification. These functions usually yield a number which is a measure of the "goodness" of the board. For most purposes, effective use of this information can be made if the goodness is classified into one of a small number of categories. One of the first forms proposed for the evaluation functions was a linear polynomial of attributes of the board, where both the attributes and their weights were chosen in consultation with domain experts. Later, in order to take into account interactions between the variables in the evaluation function, higher order polynomials were proposed. This of course resulted in a fairly rapid increase in the complexity of the function: if r th order interactions between the attributes were to be included, and the number of attributes is n , then the number of terms was of the order of n^r . Samuel's *signature tables* [33] provided a solution which exemplifies the use of

intermediate abstractions in classification. For the purposes of our discussion, Samuel's method can be described as follows.

- 1) Identify groups of attributes such that on the basis of domain knowledge there is reason to believe that they contribute to an intermediate abstraction that can be used to construct the desired classification, which in this case is a measure of the goodness of the board. The number of attributes in each group is kept small, and the attributes in a group may have some dependencies and interactions, in order to capture which polynomial terms were included in the more traditional evaluation functions. The abstractions typically correspond to the concepts in the task domain, e.g., in chess, "defensibility of king" and "material advantage" may be such intermediate concepts, each of which can be estimated by a small subset of board attributes, while the final decision about the goodness of a board configuration may be made in terms of these intermediate abstractions.
- 2) Find a method of classifying the desirability of these intermediate concepts into a small number of categories from the values of the attributes in each group. The exact method for this classification is not especially important here, though Samuel proposed a specific mechanism for it. The essence of his mechanism is a mapping from a multidimensional vector, each component of which can only take on one of a small number of distinct values, to a symbolic abstraction, which can also take on one of a small number of distinct values. This mapping may be performed by a simple table look-up for example.
- 3) The outputs of the classifiers for each group can themselves be thought of as qualitative attributes at the next level of abstraction. These attributes can be then grouped and abstracted into higher level concepts, and the process repeated as many times as necessary, with only a small number of attributes in a group at any level, until the top-level concept is a classification of the "goodness" of the board.

Let n denote the total number of attributes at the lowest level of abstraction. Let us assume that the number of attributes in each group at any level in the hierarchy of abstractions is less than some small constant upper bound n_0 (an assumption allowed in the signature table method), and further, that the groups of attributes at any level are disjoint. Then both the time and space complexities are $O(n)$ [17]. Even if a few attributes at some level are used in more than one group of attributes, which sometimes is the case, and in which case the time complexity would be somewhat worse than linear in n , clearly, the use of intermediate abstractions in classification yields substantial computational savings. Again, we are not suggesting that such conceptual decomposition of the classification process into hierarchically organized intermediate abstractions is always possible, but that, whenever possible, it is computationally advantageous to do so.

B. Hidden Units in Connectionist Networks

The computational power of using intermediate abstractions is evident from the fact that a major difference (perhaps the major difference) between modern connectionist networks and the perceptron model, is that the former provide mechanisms for capturing intermediate abstractions. In the perceptron model, since the input units were connected directly to the output unit, there was no representational mechanism to capture intermediate abstractions, and classification was performed by directly mapping input objects onto categories. Modern connectionist networks, on the other hand, contain hidden units between the input and the output units, thus providing a mechanism for representing intermediate abstractions as patterns of activity over the hidden units. The notion that the real role of the hidden units is to somehow capture these abstractions becomes clear from the following observation: in most connectionist schemes, such as the one for learning the past tenses of English language words [32], the number of hidden units in the network is critical to its performance. When the number of hidden units is too small then the problem is overconstrained and there is not enough structure to capture all the needed abstractions, as a result of which the performance of the network deteriorates markedly; and when the number of hidden units is too large then the problem is underconstrained and generalizations to the abstractions are not possible, again resulting in a marked deterioration in the network performance. One method of handling these sensitivity problems is to make the number of hidden units a parameter of the architecture, and then experiment with the value of this parameter until the number of hidden units in the network is just right.

The real computational power of modern connectionist networks is thus based on the use of intermediate abstractions, which is an important reason for the resurgence of the connectionist paradigm in AI more than a decade after Minsky and Papert had showed the inadequacies of the perceptron model. Classification in connectionist architectures is accomplished by first mapping the input entity onto classificatory abstractions, and then mapping these abstractions onto output categories. Moreover, as in Samuel's work on signature tables for game playing programs, in modern connectionist networks the intermediate abstractions can be organized hierarchically. Indeed, for large scale connectionist networks, where the number of classificatory categories and intermediate abstractions may be very large, hierarchicalization of abstractions is an important method for dealing with the complexity of learning classificatory categories and intermediate abstractions [2].

C. Symbols and Abstractions

While the intermediate abstractions are represented as patterns of activity over the hidden units in connectionist networks, there is a simpler way of capturing these abstractions: by means of discrete *Symbols*. The representation of abstractions by symbols entails a trade off between

the precision of numbers, with the concomitant problems of complexity, sensitivity, and opacity, for the simplicity, flexibility, and perspicuity of symbols. Often numbers are too precise for the task at hand, and robust symbolic hierarchical abstractions of the appropriate kind can capture almost all of the relevant information. These advantages of representing abstractions by symbols have been demonstrated most recently by Lehnert [25]. She has constructed a connectionistically inspired system, called PRO, for the task of word pronunciation, the same task that is performed by the entirely connectionist MBRtalk system. The main difference between the two approaches lies in that the PRO system uses symbols for capturing intermediate abstractions in the classification of character substrings of words. While PRO appears to perform at least as well as the MBRtalk system, it is simpler, smaller, more robust, and more perspicuous. We are not suggesting that intermediate abstractions are entirely neutral to the underlying architecture of implementation and representing abstractions symbolically is necessarily right for all tasks. Chandrasekaran *et al.* [11] provide an analysis of the interaction between the abstractions needed for problem solving and the architecture for their implementation, and suggest that connectionist schemes may be well suited for simple forms of pattern matching and data retrieval, and for low-level parameter learning. However, for capturing higher level cognitive processes the advantages of using symbols for representing abstractions are just too important.

VI. USE OF RELATIONS BETWEEN SYMBOLS FOR CLASSIFICATION

After about a decade of work on statistical classification in the pattern recognition paradigm, during which work on classification in the perceptron and the symbolic paradigms was going on roughly in parallel, Narasimhan [29] proposed a *syntactic approach* to pattern classification. The idea was to describe categories of patterns not in terms of probability distributions in multidimensional spaces, nor in terms of intermediate abstractions that can be captured symbolically, but in terms of *relations between symbols*, much as grammatical categories are described in linguistic analysis. The idea of syntactic pattern recognition is really a special case of the more general notion of *structural relations* for describing classificatory categories. Thus, even when the idea of syntax is not appropriate—it is doubtful that the notion of a picture grammar really is as general for the domain of visual objects as it appears from a purely formal perspective—the notion of structural relations for characterizing categories may still be applicable. We note that the ability to describe a category in terms of relations is a move towards *descriptions* as the basis for category characterization.

The major research directions in pattern recognition for capturing structural relations generally were formal, i.e., they used some or the other mathematical system within which theorems about relationships between categories may be provable regarding the classification performance.

In fact, this was the major reason for the original emphasis on syntactic methods, since there was a well developed theory of formal grammars already available. This emphasis on formalisms led to two constraints: firstly, often an attempt was made to force the available formalisms to fit the pattern recognition problem, generally with unsatisfactory results; and secondly, because human classification performance was more heuristic in nature, restricted formalisms could capture the quality of human performance only fleetingly.

It is interesting to note that in connectionist schemes, also classification is based on structural relations between intermediate abstractions, even though the abstractions are represented by patterns of activity over hidden units instead of being captured symbolically. The structural relations themselves are represented by connections of various types between the hidden units. Thus, in the NETalk system, the connectionist scheme for the task of word pronunciation, classification of the input words is based on the structural relations between the nonsymbolic classificatory abstractions [34].

With the introduction of syntactic/structural relations between intermediate abstractions the progression of approaches to classification becomes

numbers \rightarrow abstractions (symbols) \rightarrow relations.

Now, if one is to use relations between symbolic attributes as the basis of category characterization, then why restrict oneself to syntactic relations? Why not bring the full power, to the extent possible or necessary, of the semantics of the classificatory categories? Asking this question prepares the way for the next step in the progression of approaches to classification.

VII. KNOWLEDGE-BASED APPROACHES TO CLASSIFICATION

It is clear that each AI paradigm emphasizes different issues and poses them in a different language. e.g., the pattern recognition paradigm raises issues such as those of discriminant functions, probability distributions, and error rates, while the connectionist paradigm raises issues such as those of weights of connections, hidden units, and parameter learning. Similarly, the knowledge-based reasoning paradigm focuses on the issues of how to represent knowledge in symbolic form, how to organize and access this knowledge, how to use this knowledge for solving problems, and how to control the problem solving process. The knowledge-based approaches to the classification task attempt to answer these questions for classificatory problem solving. In this section, we will describe hierarchical classification [6], [20] as an example of knowledge-based approaches to classification, using the task domain of medical diagnosis for illustration.

A. Hierarchical Classification

In hierarchical classification, domain knowledge is organized as a hierarchical collection of categories, each of which has knowledge that helps it determine its relevance

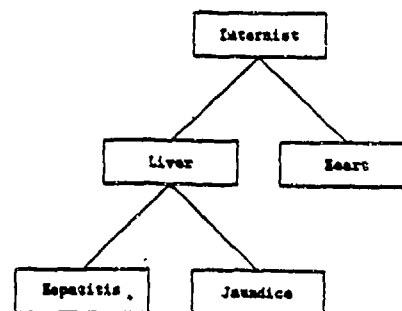


Fig. 1. Fragment of diagnostic classification hierarchy.

to the input case of unknown classification. A fragment of the classification hierarchy for medical diagnosis might be as shown in Fig. 1. Each category in the diagnostic classification hierarchy is a diagnostic concept of potential relevance to the case at hand. More general concepts (e.g., LIVER) are higher in the hierarchy, while more particular ones (e.g., HEPATITIS) are lower in the structure.

The total diagnostic knowledge is distributed over the conceptual categories in the hierarchy. Each concept has "how-to" knowledge for simple evidential reasoning in the form of several clusters of diagnostic rules: confirmatory rules, exclusionary rules, and perhaps some recommendation rules. These production rules are of the form: $\langle \text{pattern} \rangle \rightarrow \langle \text{evidence} \rangle$, e.g., "If the value of SGOT is high then add n units of evidence in favor of cholestasis," where n is some small integer. The number of rules in any one cluster is kept small, and the evidence for confirmation and exclusion is suitably weighted and combined to arrive at a conclusion to establish or reject the relevance of the category to the case, or perhaps to suspend the decisionmaking if there is not sufficient data to make a decision at the present time. The recommendation rules are optimization devices whose discussion is not necessary for our current purpose. What is important here is that when a concept in the classification hierarchy is properly invoked, a small, body of knowledge relevant for decisionmaking comes into play.

The control problem in hierarchical classification can be stated as "which conceptual category should be considered at what point in the problem solving?" In general, we would like to use domain knowledge to achieve computational efficiency by considering only a subset of all categories. Similarly, we would like to consider categories that are more promising ahead of others. The control regime natural to hierarchical classification is top-down and can be characterized as establish-refine. Starting from the root node, each concept first uses its knowledge to establish or reject itself for relevance to the entity to be classified. If it succeeds in establishing itself, then it attempts refinement by sending messages to its subconcepts who repeat the establish-refine process. If, on the other hand, the concept rejects itself, then all its subconcepts are automatically ruled out leading to a pruning of the hierarchy. The idea is to establish a conceptual category, as specific as possible, that is relevant to the input entity. Let us consider the case of a patient suffering from hepatitis as an example. Given data about this patient, first INTERNIST would establish

that there is in fact a disease, and send messages to LIVER and HEART for refinement as shown in Fig. 1. Then LIVER would establish that the disease is a liver disease, and send messages to HEPATITIS and JAUNDICE for refinement, while HEART would reject the hypothesis that the patient is suffering from a heart disease. Next, HEPATITIS would establish the disease as hepatitis while JAUNDICE would rule out the hypothesis that the disease is jaundice. Thus each concept makes decisions about its relevance to the patient data in the context of the decisions made by its superconcepts. Sticklen *et al.* [38] discuss the control issues in classificatory diagnosis in detail.

The problem solving in this approach to classification is distributed. The conceptual structures in the hierarchy are not a static collection of knowledge; instead, they are active problem-solving agents. Each of them has knowledge only about establishing or rejecting the relevance of a conceptual category, and communicates with others by passing messages. The entire ensemble of these semi-autonomous problem solving agents cooperates to perform the classification task. Goel *et al.* [19] have shown how the concurrency inherent in hierarchical classification can be exploited on a distributed memory, message passing architecture.

We note that hard probability numbers are nowhere used in diagnosis by hierarchical classification; what each problem solving agent computes are *qualitative* belief measures: "definitely present," "likely present," "definitely absent." Moreover, the computation of the qualitative values is *localized* rather than based on some global probability calculus; each agent computes the qualitative measure for its concept using only its own knowledge but in the context of its superconcepts. Medical diagnosis appears to be an instance of the class of problems in which numerical approaches, such as statistical pattern recognition, would have significant computational problems. In addition, it would pose considerable difficulty in acquiring knowledge in terms of probability distributions, at least for problems of large degree of complexity, while knowledge in the form required by hierarchical classification is often directly available from domain experts.

At our research laboratory we have used the hierarchical classification methodology to construct MDX [6], [8], [20], a medical diagnostic system for a class of liver diseases in internal medicine. The number of state variables, such as symptoms, signs, and laboratory values, describing a typical case that MDX can handle is in the hundreds, and the number of distinct conceptual categories in its diagnostic hierarchy is also close to hundred. MDX is a complex system that has been tested on a number of real world cases with a high match between its conclusions and that of human specialists. Recently, a more sophisticated version of the MDX system, called MDX2 [39], has been constructed in our laboratory.

Several concerns ought to be noted before using the hierarchical classification methodology to build knowledge used classificatory problem solvers.

- 1) Not all classification problems are necessarily solved as hierarchical classification problems. Hierarchical

classification requires that concepts in the task domain be available at several different levels of abstraction. While there are many real world domains that do satisfy this condition, not every domain need have this characteristic. There are other systems that perform classification, but without using the hierarchical point of view [1]. However, it may be better to use hierarchical classification whenever possible for reasons of computational efficiency. Let m be the number of categories at the leaf nodes of the classification hierarchy. Since the desired classification generally is one of these m categories, the time complexity of nonhierarchical classification is $O(m, t)$, where t is the time complexity of finding the relevance of a single category to the entity of unknown classification. If the number of state variables is n , and single category classification is performed using the signature table approach discussed earlier, then t is $O(n)$. In case of hierarchical classification, in the best case when all but one branch at each node in the hierarchy are ruled out, the time complexity is $O(\log(m), t)$; and in the worst case, when every branch at each node is traversed, the time complexity is $O(m, t)$. Goel *et al.* [17] provide details of the complexity calculations for classificatory reasoning. It is clear, however, that even in the worst case, the complexity of hierarchical classification is no worse than the complexity of nonhierarchical classification, and the choice between them really depends on whether it is possible to construct a classification hierarchy in the task domain of interest.

- 2) The entity to be classified may have several leaf node categories simultaneously relevant to it, rather than just one leaf node category. In medical diagnosis, e.g., a patient may have both "cirrhosis" and "portal hypertension" (which in the domain of liver diseases might be two of the leaf nodes in the classification hierarchy), and in addition, the two diseases may be causally related. Such a situation is not uncommon in other domains as well, e.g., in character recognition, the pattern to be classified may consist of be two characters touching each other rather than one single character. The hierarchical classification framework clearly can deal with such situations.
- 3) The classification hierarchy may be a tangled hierarchy, i.e., some concepts in the hierarchy may have more than one superconcept. Such a hierarchy may be untangled in the hierarchical classification framework by storing a copy of the concept in each tangled branch. This introduces redundancy in the storage of domain knowledge by the classification agent.
- 4) In general, multiple classification hierarchies may exist in the task domain, e.g., in medical diagnosis there may be one classification hierarchy for infectious diseases, and another for liver diseases. In addition, the same category may exist in more than one classification hierarchy, e.g., viral hepatitis is a conceptual category in the infectious disease hierarchy as

well as in the liver disease hierarchy. This involves coordination among the classifications reached by the different classification modules. The MDX2 system contains several classification hierarchies, and provides a mechanism for handling such interactions between them.

- 5) The problem task may require not only classification of entities onto categories, but other problem solving types as well, e.g., the diagnostic task often is functionally decomposable into the generic tasks of knowledge-directed data abstraction, and abductive assembly of explanatory hypotheses in addition to that of classification [9], [10]. This involves coordinating the actions of various problem-solving modules performing different generic tasks and cooperatively solving the diagnostic problem. The MDX system [8] contained modules for hierarchical classification and knowledge-directed data abstraction and provided mechanisms for communication between them. The MDX2 system [39] contains modules for knowledge-directed data abstraction and abductive assembly of explanatory hypotheses in addition to several hierarchical classification models, and provides mechanisms for handling interactions between them.
- 6) The conceptual structure mechanism used in hierarchical classification is only one of the several possible methods for determining the relevance of a specific category to the entity of unknown classification. In the DART system [16], e.g., the decision about the match of the category to the input data is done by using theorem-proving techniques. Alternatively, the classification category agents may make their decisions based on a causal knowledge of the domain [35]. The MDX2 systems uses such causal knowledge to derive the conceptual structure needed for category classification. In simple cases, it may be possible to use statistical pattern recognition methods for this purpose. Connectionist networks may be especially appropriate for the pattern matching operations required in simple evidential reasoning [11]. The point is that how the hypotheses are evaluated is somewhat independent of the flow of control for the classificatory task as such, even though for complex problems, a rich knowledge structure will be called for to make the decision about how well a specific category matches the data for the case in hand.

VIII. CONCLUSION

We have noted that classification appears to be an ubiquitous information processing task underlying human thought processes. The reason for this is the significant computational advantages that arise from indexing stored action knowledge over *equivalence classes of the states of the world* rather than over the states of the world themselves. We have taken the reader through a progression of approaches to classification:

numbers \rightarrow abstractions (symbols)

\rightarrow relations \rightarrow knowledge structures.

Each stage in this progression gave added power in controlling computational complexity by matching the structure of the classifier to that of the task. At the knowledge level, the computational power comes from task-specific control regimes controlling access to appropriate chunks of domain knowledge. We motivated the discussion by using classificatory diagnosis as an example in various places, but the ideas are applicable more generally.

This paper can be viewed as a bridge-building activity between three research paradigms in AI: knowledge-based reasoning, pattern recognition, and connectionism. Classification has been a major concern in pattern recognition, and an important task performed by many knowledge-based systems as well as by most connectionist networks. Thus, the classification task provides a good place to understand some of the distinctions between the three research paradigms. For well-constrained classification problems with relatively small number of categories, the numerical functions and measures used in pattern recognition models and connectionist networks typically can provide powerful classifiers which often outperform human experts by extracting the last trace of information that discrete symbolic processes can only approximate. On the other hand for complex problems involving many variables and categories, the symbolic knowledge-based approach trades off the optimality of the best functions in pattern recognition and in connectionism for computational tractability and better matching with human knowledge in the task domain. Our own research lies in the knowledge-based reasoning paradigm. Our approach has been to identify generic tasks other than that of classification, but with the similar characteristic of being a building block for intelligence. Chandrasekaran [7], [9], [10] provides an account of the repertoire of generic tasks that we have identified so far.

Many of the points made in this paper transcend the particular task of classification. In that sense, this paper can be thought of as an attempt to show the need for the emergence of symbolic structures for complex information processing transformations on representations. Cybernetics showed the power and usefulness of feedback and stability in understanding many control and communication problems. However, classical control theory is expressed in terms of numerical measures and functions. Learning and control in this framework involves parameter modification and signal propagation. The space over which parametric changes and numerical signals can provide control is quite limited. Symbolic models of the world provide greater leverage for change and control and still keep computational costs under control. Thus in biological information processing, symbolization seems to have occurred very early in evolution; Lettvin *et al.* [26] provide an account of how the early visual processing of the frog is symbolic. Once symbols were available as the language in which to perform information processing, thought eventually evolved into more and more complex symbol structures. Thus the discussion in this paper can be viewed as an intuitive account of the emergence and power of symbolic structures for complex information processing activities.

ACKNOWLEDGMENT

We are thankful to Dean Allemang for his comments on an earlier version of this paper.

REFERENCES

- [1] J. S. Aikins, "Prototypical knowledge for expert systems," *Artificial Intell.*, vol. 20, no. 2, pp. 163-210, 1983.
- [2] D. H. Ballard, "Modular learning in neural networks," in *Proc. Sixth Nat. Conf. Artificial Intell.*, 1987, pp. 279-284.
- [3] J. Bennet and R. Englemore, "SACON: A knowledge-based consultant for structural analysis," in *Proc. Sixth Int. Joint Conf. Artificial Intell.*, 1979, pp. 47-49.
- [4] B. G. Buchanan and E. A. Feigenbaum, "Dendral and meta-dendral: Their applications dimension," *Artificial Intell.*, vol. 11, nos. 1-2, pp. 5-24, 1978.
- [5] D. C. Brown and B. Chandrasekaran, "Knowledge and control for a mechanical design expert system," *IEEE Comput.*, vol. 19, no. 7, pp. 92-100, 1986.
- [6] B. Chandrasekaran, S. Mittal, F. Gomez, and J. W. Smith, "An approach to medical diagnosis based on conceptual structures," in *Proc. Sixth Int. Joint Conf. Artificial Intell.*, 1979, pp. 134-142.
- [7] B. Chandrasekaran, "Towards a taxonomy of problem-solving types," *AI Magazine*, vol. 4, no. 1, pp. 9-17, 1983.
- [8] B. Chandrasekaran and S. Mittal, "Conceptual representation of medical knowledge for diagnosis by computer: MDX and related systems," in *Advances in Computers*, M. Yovits, Ed. New York: Academic Press, 1983, pp. 217-293.
- [9] B. Chandrasekaran, "Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design," *IEEE Expert Magazine*, vol. 1, no. 3, pp. 23-30, 1986.
- [10] —, "Towards a functional architecture for intelligence based on generic information processing tasks," in *Proc. Tenth Joint Conf. Artificial Intell.*, 1987, pp. 1183-1192.
- [11] B. Chandrasekaran, A. Goel, and D. Allemang, "Connectionism and information processing abstractions: The message still counts more than the medium," to appear in *AI Magazine*, 1988.
- [12] W. J. Clancey, "Heuristic classification," *Artificial Intell.*, vol. 27, no. 3, pp. 289-350, 1985.
- [13] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: John Wiley, 1973.
- [14] R. O. Duda, P. E. Hart, P. Barret, J. Gasching, K. Konolige, and R. Reboh, "Development of the prospector consultation system for mineral exploration," SRI International Menlo Park, CA, technical report, 1979.
- [15] J. A. Feldman and D. H. Ballard, "Connectionist models and their properties," *Cognitive Sci.*, vol. 6, pp. 205-254, 1982.
- [16] M. R. Genesereth, "Diagnosis using hierarchical design models," in *Proc. Second Nat. Conf. Artificial Intell.*, 1982, pp. 278-283.
- [17] A. Goel, N. Soundararajan, and B. Chandrasekaran, "Complexity in classificatory reasoning," in *Proc. Sixth Nat. Conf. Artificial Intell.*, 1987, pp. 421-425.
- [18] A. Goel, B. Chandrasekaran, and D. Sylvan, "JESSE: An information processing model of political decision making," in *Proc. Third Expert Syst. in Gov. Conf.*, 1987, pp. 78-87.
- [19] A. Goel, J. R. Josephson, and P. Sadayappan, "Concurrency in abductive reasoning," in *Proc. DARPA Knowledge-based Syst. Workshop*, 1987, pp. 86-92.
- [20] F. Gomez and B. Chandrasekaran, "Knowledge organization and distribution for medical diagnosis," *IEEE Trans. Syst. Man Cybern.*, vol. 12, no. 1, pp. 34-42, Jan. 1981.
- [21] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biological Cybern.*, vol. 52, pp. 141-152, 1985.
- [22] J. R. Josephson, B. Chandrasekaran, J. W. Smith, and M. C. Tanner, "A mechanism for forming composite explanatory hypotheses," *IEEE Trans. Syst. Man Cybern.*, vol. 17, no. 3, pp. 445-454, May/June 1987.
- [23] L. Kanal and B. Chandrasekaran, "Recognition, machine recognition, and statistical approaches," in *Methodologies of Pattern Recognition*. New York: Academic Press, 1969, pp. 317-332.
- [24] L. Kanal and B. Chandrasekaran, "On linguistic, statistical, and mixed patterns for pattern recognition," in *Frontiers of Pattern Recognition*. New York: Academic Press, 1972, pp. 163-192.
- [25] W. Lehnert, "Case-based problem solving with a large knowledge base of learned cases," in *Proc. Sixth Nat. Conf. Artificial Intell.*, 1987, pp. 301-306.
- [26] J. Lettvin, H. Maturana, H. W. S. McCulloch, and W. Pitts, "What the frog's eye tells the frog's brain," *Proc. IRE*, 1959, vol. 47, pp. 1940-1951.
- [27] J. McDermott, "R1: A rule-based configurator of computer systems," *Artificial Intell.*, vol. 19, no. 1, pp. 39-88, 1982.
- [28] M. Minsky and S. Papert, *Perceptrons*, Expanded Edition. Cambridge, MA: MIT Press, 1988.
- [29] R. Narasimhan, "Labeling schemata and syntactic description of pictures," *Inform. and Contr.*, vol. 7, pp. 151-179, 1964.
- [30] H. W. Pople, "Heuristic methods for imposing structure on ill-structured problems," in *Artificial Intelligence in Medicine*, P. Szolovits, Ed. Boulder, CO: Westview Press, 1982, pp. 119-190.
- [31] F. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan Books, 1962.
- [32] D. E. Rumelhart and J. L. McClelland, "On learning the past tenses of English verbs," in *Parallel Distributed Processing, Vol. 1*, Rumelhart, McClelland and the PDP Research Group, Eds. Cambridge, MA: MIT Press, 1986.
- [33] A. L. Samuel, "Some studies in machine learning using the game of checkers II: Recent progress," *IBM J. of Res. Devel.*, vol. 11, no. 3, pp. 601-617, 1967.
- [34] T. J. Sejnowski and C. R. Rosenberg, "NETtalk: A parallel network that learns to read aloud," *Elect. Engng. and Comp. Sci. Dept.*, Johns Hopkins Univ., Baltimore, MD, tech. rep., 1986.
- [35] V. Seshubugamoorthy and B. Chandrasekaran, "Functional representation of devices and compilation of diagnostic problem solving systems," in *Experience, Memory, Reasoning*, J. Kolodner and C. Reisbeck, Eds. Hillsdale, N.J.: Lawrence Erlbaum, 1986, pp. 47-73.
- [36] E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*. New York: Elsevier/North-Holland, 1976.
- [37] R. R. Sokal and P. H. A. Sneath, *Principles of Numerical Taxonomy*. San Francisco: Freeman, 1963.
- [38] J. Sticklen, B. Chandrasekaran, and J. R. Josephson, "Control issues in classificatory diagnosis," in *Proc. Ninth Int. Joint Conf. on Artificial Intell.*, 1985, pp. 300-306.
- [39] J. Sticklen, "MDX2: An integrated medical diagnostic system," Ph.D. Dissertation, Dept. Comput. Inform. Sci., Ohio State Univ., 1987.
- [40] P. Szolovits and S. G. Pauker, "Categorical and probabilistic reasoning in medical diagnosis," *Artificial Intell.*, vol. 11, no. 1.2, pp. 115-144, Aug. 1978.



B. Chandrasekaran (M'67-SM'79-F'86) received his bachelor of engineering degree with honors in 1963 from Madras University, India, and his Ph.D. in 1967 from the University of Pennsylvania, Philadelphia.

He was a research scientist with the Philco-Ford Corporation, Bluebell, PA, working on speech- and character-recognition machines from 1967 to 1969. He has been at Ohio State University, Columbus, OH, since 1969. He is currently a professor of computer and information science

and he directs the artificial intelligence group.

Dr. Chandrasekaran's research activities are currently in knowledge-based reasoning. He is associate editor for artificial intelligence of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS and chairs the society's Technical Committee on artificial intelligence.



Ashok Goel studied physics as an undergraduate in India. He obtained his Masters in physics in 1980 and his Masters in computer and information science in 1982—both from the Ohio State University.

He is presently a doctoral candidate in computer and information science and a research associate with the Laboratory for Artificial Intelligence at the Ohio State University. His current research interests include knowledge-based reasoning, naive physics, and neural networks.

Appendix F

A Mechanism for Forming Composite Explanatory Hypotheses

A Mechanism for Forming Composite Explanatory Hypotheses

John R. Josephson, B. Chandrasekaran,
Jack W. Smith Jr., and Michael C. Tanner

Laboratory for AI Research (LAIR)
Department of Computer and Information Science, and
Laboratory for Knowledge Based Medical Systems
Department of Pathology
The Ohio State University
Columbus, Ohio 43210

Contact: John R. Josephson
Ohio State University
LAIR, CIS Dept.
228 CAE Bldg.
Columbus, Ohio 43210-1277

Phone: 614-292-0208
Netmail: Josephson@tut.cis.ohio-state.edu

A revised version of the article appearing in *IEEE Transactions on Systems, Man, and Cybernetics*,
Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning, May/June, 1987, pp. 445-454.

A Mechanism for Forming Composite Explanatory Hypotheses^a

John R. Josephson, B. Chandrasekaran,
Jack W. Smith Jr., and Michael C. Tanner

Abstract

We describe a general problem solving mechanism that is especially suited for performing a particular form of *abductive inference*, or best explanation finding. A problem solver embodying this mechanism *synthesizes composite hypotheses* by combining simple hypotheses to satisfy explanatory goals. These simple hypotheses are formed by instantiating prestored explanatory "concepts". In this way the problem solver is able to arrive at complex, integrated conclusions which are not pre-stored. We present a computationally-feasible, task-specific problem-solving mechanism for a particular information processing task which is nevertheless of very great generality. The task is that of synthesizing coherent composite explanatory hypotheses based upon a prestored, and possibly vast collection of hypothesis-generating concepts. This is seemingly a common task of intelligence, and potentially a major component of diagnostic reasoning, especially where single-fault assumptions are inappropriate. This work contributes to showing how it is computationally possible to come to "know" based upon the evidence of the case.

In this paper we describe the mechanism both functionally and structurally; that is, the *why* and *what* of the main computations are described, together with algorithms that show *how* each of these computations can be accomplished. The mechanism integrates a classification machine, used for selecting plausible hypotheses, with a specialized means-ends machine, used for assembling a best explanation from the plausible hypotheses thus selected and for pointedly investigating alternative explanations. There are also two other specialized mechanisms for the subsidiary functions of: recognizing the applicability of a hypothesis to the situation, and of interpreting the situation-specific raw data to satisfy the informational needs of the other components. The result of combining these distinct computational mechanisms is an integrated knowledge-based problem solver, functionally suited to its abstract information processing task.

^aThe present paper is an expanded and revised version of "Abduction by Classification and Assembly" which was presented at The Philosophy of Science Association Biennial Meeting for 1986 and appears in PSA 1986, Volume One.

Although the mechanism we describe here is abstracted from the architecture of the Red-2 system, several other diagnostic AI systems embody it too, in varying degrees.

1. Introduction

1.1. Methodology

Artificial Intelligence is the study of complex information-processing problems that often have their roots in some aspect of biological information processing. The goal of the subject is to identify interesting and solvable information processing problems, and solve them.¹

Vision is an information processing task, and like any other, it needs understanding at two levels. The first, which I call the computational theory of an information processing task, is concerned with what is being computed and why; and the second level, that at which particular algorithms are designed, with how the computation is to be carried out.²

-- David Marr

Cognitive problem solving too can be understood in terms of recognizable information processing tasks subject to the same two-level understanding that is suggested for vision. The computational theory of a cognitive task is an understanding of *what* is being computed (input/output), and *why* (what its significance is); that is, an understanding of the *function* or *goal* of the computation. The second level is an understanding of *how* the computation can be carried out by particular algorithms or mechanisms. In this paper we describe one such task, a form of abductive inference. We show how it can be accomplished under appropriate circumstances by way of certain subtasks. We describe the major subtasks both functionally, and in terms of mechanisms whereby they can be efficiently accomplished.

The work reported here takes place in the context of a theory of generic tasks in knowledge-based problem solving³. The theory proposes that complex reasoning processes can be analyzed in terms of a small set of *basic types of reasoning* (the generic tasks) each of which corresponds to an information processing strategy especially suited for achieving a particular knowledge-level functionality. Thus for example hierarchically organized symbolic pattern matching is a particularly apt strategy for concept matching, that is, for matching a prestored concept to a situation to determine whether the concept applies to the situation. Generic tasks may be thought of as "types of problem solving⁴", or as "primitive abilities" providing "building blocks of intelligence".

In our laboratory at Ohio State we have developed a software tool for each of the tasks that we have identified so far, and an integrated toolset is presently under development. The generic tasks include: hierarchical classification^{5, 6} plan selection and refinement^{7, 8, 9} (for routine design and planning),

concept matching^{5, 3} knowledge-directed indirect inference¹⁰ (for intelligent data abstraction and retrieval), prediction by abstracting state changes¹¹, and assembly and criticism of composite explanatory hypotheses¹².

1.2. Scope

The mechanism described in this paper is that of a composite problem solver which arrives at abductive conclusions by using hierarchical classification, concept matching, knowledge-directed indirect inference, and assembly and criticism of composite explanatory hypotheses. A classification machine, used as a source of plausible hypotheses, is united with a specialized means-ends machine, which is used for assembling an overall best explanation from the plausible hypotheses, and also for criticizing the hypothesis by pointedly investigating the space of alternative hypothesis assemblies. There are also two other specialized mechanisms for the subsidiary functions of: recognizing the applicability of an explanatory concept to the situation, and for interpreting the situation-specific raw data to satisfy the information needs of the other components. The result of putting together these distinct computational mechanisms is an integrated knowledge-based problem solver, functionally suited to the abstract information processing task of forming composite best explanations, based on prestored explanatory concepts.

The present paper will develop the idea of abductive inference, and describe the mechanism and its rationale in some detail. The emphasis will be on the assembly and criticism of composite hypotheses, and on classification as a mechanism for accessing explanatory concepts.

1.3. Red

Red is a knowledge-based medical expert system for use in blood banks as a red-cell antibody identification consultant.^{13, 14, 15} The system has now been through two working versions, and a third is under construction at the time of this writing. The present paper presents an abstract description of the problem solving mechanism of Red-2, the second distinct version of the system. Thus Red-2 serves as a working proof of the realizability of most of the abstract design. An evaluation of Red-2's performance has been made¹⁶, and shows that the system almost always produces clinically acceptable answers, even in complex cases.

2. Abduction

Abduction or Inference to the Best Explanation is a form of inference that follows a pattern something like this:

D is a collection of data (facts, observations, givens),
H explains D (would, if true, explain D),
No other hypothesis explains D as well as H does.

Therefore, H is probably true.

The strength of an abductive conclusion will in general depend on several factors, including:

- how good H is by itself, independently of considering the alternatives,
- how decisively H surpasses the alternatives,
- how thorough the search was for alternative explanations, and
- pragmatic considerations, including
 - the costs of being wrong and the benefits of being right,
 - how strong the need is to come to a conclusion at all, especially considering the possibility of seeking further evidence before deciding.

Abductions, as we have just characterized them, go from data describing something to an explanatory hypothesis that best accounts for that data.

Notice that calling an inference "abduction" carries with it the idea of its goal: a best explanation. Contrast this with characterizing an inference as "deduction", which carries instead the idea of a constraint that is satisfied: that the inference is guaranteed to be truth-preserving. Since there is no intrinsic incompatibility between explanatory goals and truth-preservation constraints, it is conceivable for there to be *deductive abductions*. In fact, if all of the alternative ways of explaining something are exhaustively enumerated, and all but one of the explanations are decisively eliminated, the overall pattern of inference is deductively valid.

'There is no great mystery in this matter', he said, taking the cup of tea which I had poured out for him; 'the facts appear to admit of only one interpretation.'¹⁷ -- Sherlock Holmes

Even when they are not deductively valid, abductions, besides being intuitively appealing, can be seen to carry logical force. When we have come up with all of the "plausible" explanations we can find for some body of data, and have found compelling, if not decisive, evidence against all but one, then we have good reasons for accepting that one best explanation. Reasons against the other alternatives have been transformed into reasons for accepting the one. Whether we suspend judgment, or go ahead and accept the indicated explanation with some particular degree of confidence, should properly depend on the factors we have enumerated above.

C. S. Peirce used the term "abduction" for a form of inference close to what we describe here¹⁸. Gilbert Harman and others have written of "inference to the best explanation" for essentially the same pattern^{19, 20, 21}; and Lycan calls it "the explanatory inference"²². Sometimes a distinction has been made between an initial process of coming up with explanatorily useful hypothesis alternatives, and a subsequent process of critical acceptance where a decision is made as to which explanation is best. Often the term "abduction" has been reserved for the initial, hypothesis-originating stage¹⁸. We use the term here for the whole process of inferring from the data to the best explanation.

John McCarthy has proposed an inference form called "circumscription" to express the logical leap of assuming that all the objects I know about with property P, are in fact all of the objects that exist which have property P.²³ The leap of assuming (in effect) that all of the plausible explanations we can find, are in fact all of the plausible explanations, can be seen as an application of circumscription. Since circumscription is obviously not a logical axiom (sometimes a circumscription is a reasonable step to take, and sometimes not) it must be that some factors of the situation determine whether a step which has the form of a circumscription is reasonable or intelligent. We have identified above a number of the factors relevant to a presumption that we have covered for all plausible explanations.

Arguably abduction is itself an epistemologically fundamental form of reasoning, not reducible to deduction, probabilistic induction, or any combination of them.^{19, 24}.

Whether or not abductions can be justified on logical grounds, they appear ubiquitous in the unselfconscious reasonings, interpretations, and perceivings of ordinary life, and in the more critically self aware reasonings upon which scientific theories are based²⁵. It is a common view that *diagnostic reasoning in general is abduction*^{26, 27, 28}. The idea is that the task of a diagnostic reasoner is to come up with a best explanation for the symptoms, findings for the case which show abnormal values. The explanatory hypotheses appropriate for diagnosis are malfunction hypotheses - typically disease hypotheses for physicians, and broken-part hypotheses for mechanical systems.

The characteristic reasoning processes of fictional detectives has been characterized as abduction²⁹. It has been alleged that there are at least 217 abductions to be found in the Sherlock Holmes canon³⁰.

The following example is offered in the spirit of showing that abductive reasoning is quite ordinary and commonsensical.

Joe: Why are you pulling into the filling station?
 Tidmarsh: Because the gas tank is nearly empty.
 Joe: What makes you think so?
 Tidmarsh: Because the gas gauge indicates nearly empty. Also I have no reason to think that the gauge is broken, and it has been a long time since I filled up the tank.

Under the circumstances, the nearly empty tank is the best available explanation for the gauge indication. Tidmarsh's other remarks can be understood as being directed to ruling out a possible competing explanation (broken gauge) and supporting the plausibility of the preferred explanation.

3. Organizing Concepts, Assembling Hypotheses

In some problem situations abduction can be accomplished by a relatively simple matching mechanism, with no particular concern taken for focusing the control of which explanatory concept to consider, or for controlling the assembly of composite explanations. For example, if there are only a small number of potentially applicable explanatory concepts, and if time and other computational resources are sufficiently abundant, then each hypothesis can be considered explicitly. Of course it is a pretty limited intelligence that can afford to try out each of its ideas explicitly on each situation. The classification mechanism we describe here can be seen as meeting the need for organizing the prestored explanatory concepts, and for controlling access to them. It provides a good mechanism for the job whenever knowledge is available of the right sort to make it go.

If the number of potentially applicable hypotheses is at all large, and if more than one can be correct at the same time, then the combinatorics of the situation will not permit us to have one pre-established pattern for each possible conclusion. One main alternative is to actively construct the abductive conclusion as a combination of sub-hypotheses. Up to 2^n different combined conclusions are made available by assembling from a space of n possible hypotheses. Thus a very large space of possible conclusions can result from a relatively small space of primitive categories. For example the Red-2 system has 54 most-detailed hypothesis categories, giving rise to 2^{54} or more than 10^{16} potential conclusions. (Many of these, however, would not be internally consistent, and so would never be produced by the system. Eliminating inconsistent conclusions still leaves more than 10^{12} possible conclusions.) The mechanism we describe here is capable of efficiently picking out the best combination, even from so large a space.

However access to plausibly applicable concepts is controlled, if knowledge is available so that an

accurate confidence evaluation can be made for each concept *in isolation from all of the others*, then a composite abductive conclusion can be simply formed as the conjunction of all those concepts that match above some threshold of confidence. Yet this sort of decisive recognition knowledge is rarely available. At the time that knowledge is compiled for a concept, knowledge for decisive confirmation (so called "pathognomonic evidence") may be unavailable, so that the very strongest that can be determined based on direct evidence is that it is "very likely" that the concept applies to the situation. Psychiatric diagnoses probably tend to be of this sort. More significantly, *at run time* there will often be too little actual data about the situation to rule-in and rule-out decisively based on local match, even if there are potential items that would be decisive if they were available. When the data is too weak to resolve the situation based only on local matching, taking account of the interactions between hypotheses can significantly contribute to getting more conclusion out of the data. An important non-local way to achieve a high degree of confidence is to explicitly rule out alternative ways of explaining things, so that some of the data have only one possible explanation. When this can be done a small hypothesis will stand out as a best explanation for a portion of the data, and thus be a good candidate for inclusion as part of the composite best explanation for the case. In this way a *small abduction* over a portion of the data is performed in support of the *larger abduction* necessary for solving the whole problem. Besides the role of explanatory alternatives, hypotheses may interact in a number of other ways bearing on acceptance, some of which we will discuss later on. In general our explanatory concepts rarely have the kind of independence required for completely separate evaluation to be a viable method, and some form of active control over the formation of composites will be necessary.

4. The Mechanism

The overall function of an abduction machine can be described as that of producing a "best explanation" for a given set of data. An important side effect should be that information is made available about where there are alternative ways of explaining things, so that this information can be used for critically assessing that proposed best explanation, deciding where and how far to trust it.

We present here a functional description of a particular abstract abduction machine, which we may as well call "the Red inference engine" in honor of the system from which it is abstracted. We include enough detail about the major algorithms to make it clear how they work.

4.1. Task and Subtasks

Suppose that we intend to build a knowledge-based system to capture expertise at a certain abductive task. That is, our system is to take, as input, data of a certain type; and produce, as output, best explanations for a well-defined subset of the input data. Suppose that we are given a large number of potentially applicable hypothesis "concepts" or "frames" to base the system on; and that more than one concept can correctly apply at the same time.

Notice that this is precisely the diagnostic situation a physician must face, where the pre-enumerated hypotheses correspond to known and named diseases, and where multiple diseases are common, especially among the very sick people seen at major hospitals, and among those with unobvious ailments. Notice too that this is the situation confronted by the operator of a chemical processing or nuclear power plant, where a single original malfunction can quickly cascade into a multiple malfunction situation. Notice also that this is (an aspect of) the situation faced by *any intelligent knowledge-using agent* facing a complex, changing world, armed primarily with "concepts" of what is possible, and having the goal of trying to "understand" some part of its experience by forming a "good" composite hypothesis.

Suppose further that interactions of various sorts between the pre-enumerated hypotheses can occur, making it unsatisfactory to just match each separately to the case and accept all those above a certain threshold of confidence.

One way to organize a system for this sort of task, and indeed the organization described here, is to set up separate problem-solving structures for the distinct subtasks of:

- coming up with a relatively small number of "plausible" hypotheses from the much larger number of prestored patterns,
- building a "best" composite hypothesis using these plausible hypotheses as available parts, including testing and improving the "goodness" of the composite.

We will see that this decomposition provides a good way of controlling the potentially explosive combinatorics of the problem.

4.2. Removing Irrelevant and Unlikely Hypotheses

By setting one problem solver to filter the primitive hypotheses, letting through only those plausible for the case, we potentially make a great computational contribution to the problem of finding the best composite. By making only moderate cuts in the number, say n , of hypothesis parts to consider, we can make quite deep cuts in the 2^n composites that can be generated from them. For example if we assume

that 63 prestored hypothesis patterns get cut down to 8 which are plausible for the case at hand, we have cut the number of potentially generable composites from 2^{63} , which is the number of grains of rice on the last square of the chessboard in the classical story, or more than 9 quintillion; down to $2^8 = 256$.

The INTERNIST system³¹ (for diagnosis in internal medicine) can be viewed as doing this sort of hypothesis screening when it considers only the subset of prestored diseases that are "evoked" by the present findings. In this way it screens out those hypothesis which are completely irrelevant for explaining the findings. It cuts the number down even further when it scores the evoked diseases for confidence and only continues to consider those above a certain threshold. This can be seen as a kind of screening out of hypotheses for low likelihood of being correct, likelihood being measured primarily by quality of match to the case data.

The DENDRAL system³² (for elucidating molecular structure from mass spectrogram and other data) explicitly performs such a screening subtask. During the initial "planning" phase it uses the data provided to it to generate a "BADLIST" of molecular substructures that must not appear in the hypothesized composite structures. This BADLIST is used to constrain the search space during the subsequent enumeration of all possible molecular structures consistent with the constraints. That is, (in the present terms) DENDRAL first rules out certain hypotheses for bad match to the case data, and then generates all possible composite hypotheses consistent with not including the ruled-out ones (and other constraints). We note that DENDRAL devotes a separate problem solver, with its own knowledge structures, to the initial screening task.

In the generalized set covering model of diagnosis and abduction^{33, 34} a disease is associated with a certain set of findings that it potentially covers (i.e. that it can explain if they are present). The diagnostic task is then viewed as that of generating all possible minimum coverings for a given set of findings, by sets associated with diseases (In order to use this as a basis for further question asking). In expert systems built using this model, a match score is computed for each relevant disease each time new findings are entered for consideration, and match scores are used, when appropriate, as the basis for categorically rejecting disease hypotheses from further consideration.

In the Red inference engine a separate problem-solving structure is devoted to the hypothesis selection subtask. It runs first, as soon as the case is started up, and produces a set of hypotheses, each hypothesis being the result of matching a prestored concept to the case. The hypothesis produced are all

explicitly relevant for explaining features of the case, and many potential hypothesis do not appear, having been categorically ruled out. Each hypothesis arrives with a symbolic likelihood, the result primarily of case-specific quality of match, but also of case-independent knowledge about frequencies of occurrence. The Red engine is distinguished from INTERNIST and the set-covering systems in that it devotes a separate problem solver explicitly to the hypothesis selection subtask. The advantage of a separate problem solver is that it can be designed specifically for the generic hypothesis selection task.

4.3. Synthesizing a "Best" Composite

Synthesizing a "best" composite can turn out to be computationally expensive. If there are N plausible hypotheses, then there is a space of 2^N composites that can be made from them. If each needs to be generated separately in order to determine which is best, then things can get rapidly out of hand. Clearly it is normally preferable to adopt strategies that allow us to avoid generating all of the combinations.

Sometimes the problem might be completely dominated by the difficulty of coming up with even one good composite explanation. It can be shown that the problem of coming up with just one consistent composite hypothesis that explains everything, under conditions where many of the hypotheses are incompatible with each other, is NP complete.^b

The authors of DENDRAL saw its job as that of generating all possible composites that were allowable based upon the previously established constraints on submolecules, and the known case-independent chemical constraints on molecular structure. In contrast INTERNIST terminates (after cycles of questioning, which we ignore for these purposes) when it comes up with its single best composite. The set-covering model generates all possible composites of minimal cardinality, but avoids having to enumerate them explicitly by factoring the combinations down into disjoint sets of generators.

The Red engine first generates a single, tentative "best" composite, and then improves it by criticism and suitable adjustment. In order for the the criticism to be accomplished, certain other composite hypotheses are generated, but only a relatively small number of them. Again, the Red engine devotes a distinct problem solver to a distinct task, in this case that of forming a best composite. The initial composite hypothesis is formed to be one which explains all of the data (or that part that needs to be

^bBy reduction to 3-SATISFIABILITY³⁵

explained), which is maximally plausible, or nearly so, and internally consistent.⁶

There are often more than simply computational feasibility considerations involved in a decision to generate just the one best composite instead of generating all composites subject to some constraints. For purposes of action an intelligent agent will typically need a single best estimate as to the nature of the situation, even if it is only a guess, and does not need an enumeration of all possible things it could be. By rapidly coming to a best estimate of the situation, the agent arrives quickly at a state where it is prepared to take action. If it had to enumerate a large number of alternatives, this would not only take longer in the generation of them, it would take longer to figure out what to do next. It is difficult to figure out what to do if proposed actions must try to cover for all of the possibilities.

This is not to deny the possibility of situations where careful and intelligent reasoning requires the generation of all of the *plausible* composites (i.e. those with a significant chance of being true), so that they can all be examined and compared. This might especially be called for where the cost of making a mistake is very high, as in medicine, or where there is plenty of time to think over the alternatives. Of course generating *all* of the plausible composites will be computationally infeasible if there are a lot of plausible fragments to choose from and the situation calls for many-part solutions. Moreover, besides being a computationally expensive strategy, generating all of the alternatives will not typically be necessary, as we will often be able to compare composites *implicitly*, by comparing alternative ways of putting them together. For example in comparing little hypotheses h_1 and h_2 , we are implicitly (partially) comparing all composites containing h_1 with those containing h_2 .

4.4. Criticism: Testing and Improving the Composite

In general it is important to have some idea of how good a composite is, so that an agent can decide whether to act boldly or be cautious; for example deciding to gather more evidence before taking action. Moreover, some critical assessment is necessary, because, as we said, the appropriate confidence of an abductive conclusion depends in part upon how well it stacks up to the competition. This applies to the evaluation of composite hypotheses no less than it applies to simple ones.

For each of the composite hypotheses it constructs, DENDRAL generates a prediction of how the mass

⁶Note that the findings to be explained are in general a *proper subset* of all of the findings of the case. We might try to explain the patient's symptom, but we won't try to explain his age.

spectrogram should appear. Those hypotheses whose predictions mismatch sufficiently are excluded, and the rest are ranked based upon quality of this match. Again DENDRAL devotes a special knowledge-based problem solver to the task, though it is tuned to predictions based upon molecular fragmentation, and is not domain-independent in character.

INTERNIST and the set-covering model appear not to do anything that corresponds to this sort of criticism. Indeed INTERNIST commits itself irrevocably to each hypothesis in the growing composite before it goes on to decide on the next, and has nothing corresponding to the Red engine's tentative initial assembly. The set covering model builds in the critical criterion of simplicity in the form of a guarantee that the problem solver will produce composites with the minimum cardinality sufficient to account for all of the findings.

4.5. Major Modules

The two major modules of the Red inference engine are:

- a classification machine for selecting plausible hypotheses,
- a specialized means-ends machine for assembling a subset of the plausible hypotheses into a "best" composite explanation. The hypothesis assembler is under the control of an overview critic (described here algorithmically) which uses the assembler, first to produce a tentative initial composite, then repeatedly to explore the space of alternative composites, and then finally to build a finished "best explanation" after the pointed investigation of alternative explanations. This overview critic also does some processing to guarantee that the composite it finally produces is parsimonious, i.e., has no explanatorily superfluous parts.

It is important to note that the second of these modules is usable separately from the first one, and so, for example, some structural model of a device could be exploited in some fashion to generate malfunction hypotheses.^{36, 37, 38} What the assembler/critic needs is a source of hypotheses, each hypothesis offering to explain some portion of the data, and each evaluated to determine a degree of plausibility. The assembler/critic will also need to have access to various sorts of information about how the hypotheses interact.

4.6. The Classification Machine

Taking the MDX^{39, 40} system as its point of departure, the classifier is implemented as a taxonomic hierarchy of hypothesis specialists. Each specialist in the hierarchy specializes in a single "concept". When invoked it will match that concept to the details of the case, either ruling it out of further consideration, or else producing a hypothesis that has an associated symbolic likelihood, and offers to explain certain of the findings of the case.

The hierarchy organizes the specialists from most general at the top, to most specific at the tip nodes. The hypothesis selection activity proceeds in a top-down, more-general-to-more-refined manner, taking advantage of the search pruning effect that comes from ruling out whole subtrees of hypotheses by ruling out at high levels of generality. This top-down, prune-or-pursue control regime, associated with MDX-like diagnostic systems has been called "establish-refine". It can in principle proceed in parallel, matching of two sub-concepts being typically independent of each other. By efficiently pruning the search for plausible hypotheses, establish-refine is a significant contributor to taming the combinatorics of the problem space. It makes it efficient and practical to search a very large space of stored concepts for just those that plausibly apply to the case.

4.7. Plausible Hypotheses

Each concept that is considered and cannot be ruled out is matched against the data of the case to produce a description of which parts of the data it can explain (or contribute to explaining), and how plausible it is under the circumstances. Concept matching for plausibility has been discussed elsewhere.^{5,3} Each plausible hypothesis delivered by the classifier thus comes with:

- a description, particularized to the case, of which findings it offers to explain,
- a symbolic plausibility value representing a symbolic *prima facie* estimate of likelihood for the hypothesis.

Each plausible hypothesis has its own consistent little story to tell, and to contribute to the larger story representing the abductive conclusion.

4.8. Hypothesis Interactions

Hypothesis interactions are considered to be of two general types, each with its own kind of significance for the problem-solving:

- *explanatory interactions*, e.g., due to overlapping in what the hypotheses can account for, and
- *substantive interactions* of mutual support and incompatibility, e.g., resulting from causal or logical relations.

For example, two disease hypotheses might offer to explain the same findings without being especially compatible or incompatible causally, logically, or definitionally. On the other hand hypotheses might be mutually exclusive (e.g. because they represent distinct sub-types of the same disease), or mutually supportive (e.g. because they are causally associated). The INTERNIST system did not make a clear distinction between hypotheses which are competitors because they are both capable of explaining the same findings in the case (thus not both needed), and those that are competitors because they are

mutually exclusive.⁴¹ INTERNIST was really only concerned with the former type. In general the elements of a diagnostic differential need to be *exhaustive* of the possibilities, so that at least one must be correct, but they need not be mutually exclusive. If they are exhaustive then evidence against one of them is transformed into evidence in favor of the others.

The following types of hypothesis interaction can be accommodated and treated appropriately by the mechanism we are describing. Appropriate handling for the first four of them has been implemented and tested:

- A and B are mutually compatible, and represent explanatory alternatives where their explanatory capabilities overlap.
- Hypothesis A is a subhypothesis of B (i.e., a more detailed refinement).
- A and B are mutually incompatible.
- A and B cooperate additively where they overlap in what they can account for.
- Using A as part of an explanation suggests using B also.
- A, if it is accepted, raises explanatory questions of its own that can be resolved by appeal to B.

An example of this last type occurs when we hypothesize the presence of a certain pathophysiological state to explain certain symptoms, and then hypothesize some more remote cause to account for the pathophysiological state. The tummy ache is explained by the presence of the ulcers, and the ulcers are in turn explained by the anxiety neurosis.

4.9. The Hypothesis Assembler

A mechanism for hypotheses assembly is used which is reminiscent of the means-ends regime of GPS, The General Problem Solver.⁴² Its overall goal is to explain all of the finding that need to be explained. It detects differences between the goal state (everything explained) and the present state (the working hypothesis does not explain everything), and focuses on a salient difference (a most significant unexplained finding). It uses this unexplained finding to select a hypothesis part to integrate into the growing working composite.

We begin by describing a basic hypothesis assembler, capable only of treating one type of hypothesis interaction. Then we will describe how it can be enhanced to treat the other types of interaction appropriately.

4.9.1. The Basic Assembler

The basic assembler treats only hypotheses that are mutually compatible, and that represent explanatory alternatives where their explanatory capabilities overlap. A set of findings is given, the goal is to assemble an explanation for them, and to do so in a manner that respects the plausibilities of the candidate parts. It works by using the plausibilities to guide the means-ends search.

Procedure:

- Set the initial composite to the empty set.
- Loop until there is nothing left to explain, or nothing left that can be explained.
 - Focus attention on an unexplained finding (initially the whole set is unexplained). If domain knowledge is available to point out the most significant unexplained finding, then well and good; but if not, then the choice can be made at random.
 - Pick the most plausible hypothesis that explains that finding. If no plausible explanation for it can be found, then note the finding as unexplainable and loop again, else continue. If two or more explanations tie for maximal plausibility, choose one at random.
 - Include the newly chosen hypothesis into the set of hypotheses that constitutes the growing composite hypothesis. That is, set the new composite to the union of the old composite and the set whose member is the chosen hypothesis.
 - Compute what the composite can now explain, and determine the unexplained remainder.
- End loop.
- Return the value of the composite.

The basic assembler produces a composite hypothesis which is as complete as possible. Since it uses the most plausible explanatory hypothesis at each choice point, the composite hypothesis is maximally plausible as well, or nearly so.^d

It is easy and computationally inexpensive to rid the composite of explanatorily superfluous parts.^e This can be done after the composite is built, or else parsimony can be enforced as the assembly proceeds. We thus arrive at a composite hypothesis which is *as complete as possible*, *maximally plausible* (or nearly), and *parsimonious*.

^dThe conditions under which this process produces an optimally plausible composite have been investigated, and will possibly form the subject of a future paper. Yet in general this mechanism will deliver the "correct answer" if one stands out on the basis of the evidence, and will fail only if the evidence of the case is weak or erroneous, or if knowledge in the system is wrong or missing. Thus guarantees of maximal plausibility for the composite are not really very significant. Intelligent agents do not need to be especially good at making forced choices between nearly equal alternatives.

^eCheck through the parts in order of least plausible to most plausible; for each part compute the explanatory capabilities with the part removed; and check to see if there is any loss.

Note that this interpretation of Occam's Razor has clear epistemic virtues. Logically the composite hypothesis is a conjunction of little hypotheses; so, if we remove one of the conjuncts the resulting hypothesis is distinctly more likely to be true, since it makes fewer commitments. Superfluous hypothesis parts make factual commitments, expose themselves to falsity, with no compensating gain in explanatory power. Thus the sense of parsimony we propose here is such that *the more parsimonious hypothesis is more likely to be true.*

Since the assembly process added monotonically to a growing hypothesis, with incrementally growing explanatory power, and with no backtracking, the process is computationally very inexpensive, at least if it is inexpensive to compute explanatory power. In general the greatest computational expense will be in checking through the available hypotheses to determine which one is the most plausible way to explain the finding of attention. But the classifier will collaborate to reduce the alternatives to a relatively small number, and one pass through the set will suffice. *Thus the whole process of assembly is computationally very efficient.*⁴³

4.9.2. Extensions and Elaborations to the Basic Assembler

Extensions can be made to the basic assembler to handle the other types of hypothesis interaction we have mentioned.

If hypotheses in the space come with subtype relationships, as they normally would with a hierarchical classifier, the assembler can preferentially pursue the goal of explanatory completeness and secondarily pursue the goal of refining the constituent hypotheses down to the level of most detail.

A more difficult problem is in devising a strategy for when some of the hypotheses in the space are mutually incompatible.^f What was done in Red-2 is to maintain the consistency of the growing hypothesis as the assembly proceeds. If a finding is encountered whose only available maximally plausible explainers are incompatible with something already present in the growing hypothesis, then one of these newly encountered hypotheses is included in the growing hypothesis, removing from the composite any parts inconsistent with the new one.^g The basic idea is that the finding must be explained, even if that

^fWe assume the ability to detect that hypotheses are incompatible.

^gIf we remove parts from the growing hypothesis we introduce the danger of an infinite loop, but fortunately this can be dealt with fairly readily. We suitably raise the standards for reintroducing a hypothesis for the second time in precisely the same situation in which it was first introduced. The second time around we require that there be no net loss of explanatory power for the whole assembly resulting from reintroducing the hypothesis and removing its contraries from the assembly. There are a variety of acceptable measures of explanatory power that will serve here to guarantee progress.

forces a serious revision of the growing hypothesis. This seems to be a rather weak way of handling incompatibles, and the authors feel that significant improvements are possible.

If hypotheses can cooperate additively where they overlap in what they can explain, all we need to do is to suitably incorporate this knowledge into the methods for computing what a composite hypothesis can explain.

In order to handle the kind of hypothesis interaction where one hypothesis suggests the use of another, as for example if there is available knowledge of a statistical association, we can give extra plausibility credit to the suggested hypothesis if the hypothesis making the suggestion is already part of the growing composite. The availability of a way to grow the hypothesis preferentially along lines of statistical association provides a rudimentary ability for it to grow along causal lines as well.

A more interesting ability to grow along causal lines results if we permit one hypothesis, if it is accepted into the growing hypothesis, to raise explanatory needs of its own. For example, a newly added hypothesis can be posted as a kind of higher-level finding which needs to be explained in its turn by the growing assembly. Thus at the same time that the newly added hypothesis succeeds in explaining some of the findings, it introduces a "loose end". This provides a way in which the growing hypothesis can move from hypotheses close to the findings of the case, and towards more and more remote causes of those findings.

4.10. The Overview Critic

Procedure:

- The assembler is invoked to produce a tentative best explanation.
- Explanatorily superfluous parts are removed.
- The assembler is invoked repeatedly as necessary to assess which of the hypotheses in the composite are indispensable. A hypothesis is judged indispensable if removing it from a composite which is a complete explanation leaves behind a composite which cannot then be assembled to completion without reintroducing the removed one. It follows that a hypothesis is indispensable if and only if something that it explains *has no other plausible explanation*. It is important to distinguish between hypothesis parts which are non-superfluous relative to some particular composite, that is they cannot be removed without explanatory loss, and indispensables without which no complete explanation can be found in the whole hypothesis space.
- The non-indispensable parts of the composite are then removed, and the assembler is invoked again to rebuild from the core of indispensables back to a complete explanation. This rebuilding process might again introduce explanatorily superfluous parts that will need to be cleaned out, but it cannot introduce any more indispensables. Since an indispensable explains something that has no other plausible explanation, every indispensable is already

present in any complete explanation.

- Any newly introduced explanatorily superfluous parts are removed.

At the end of this process the composite hypothesis explains as much as possible, is maximally plausible (or nearly so), is parsimonious, and has been built up by going from a core of hypotheses which are most certain.

At this stage the best explanation has been inferred, or at least A best explanation has been inferred, there being no *a priori* guarantee that a best explanation is unique. Under some circumstances the reasoning process will have virtually *proved* the correctness of its conclusion. If each part of the composite is indispensable (in the sense above), then the system has proved that it has produced the correct explanation, assuming that the data is correct and the knowledge base is complete. That is, the system will have proved that it has come up with the only complete and parsimonious explanation available to it. When parts of the conclusion are not indispensable, the system will have discovered that alternative explanations are possible, so appropriate cautions may be taken in using the abductive conclusion.

5. Extensions and Elaborations of the Mechanism

The degree of intimacy, and the nature of the relationships, between the classifier, various critics, and the means-ends assembler, is an unresolved research issue which we are actively exploring. In Red-2, the classifier runs first, producing a set of plausible hypotheses, and then is followed by the critic, which uses the assembler to produce the best explanation. In the future we anticipate a version where the classifier and an assembler/critic run concurrently, with the latter using its perspective on the progress of the problem solving to help guide the search for plausible hypotheses. Red-2's assembler/critic built up composite hypotheses using only tip node hypotheses delivered to it by the classifier. We anticipate that in the next version composites will first be assembled at higher levels of generality, and then refined into more detailed hypotheses using nodes lower in the hierarchy. More distantly we envision a version where lots of little hypothesis assemblers and critics are distributed over a problem solving structure that makes local abductions, producing little assembled best explanations. By solving subproblems the little abducers will serve the needs of larger abducers, and make it possible to assemble hypotheses from parts which are themselves assemblies.

6. Summary

We have described how best explanations can be inferred by a mechanism which tames the combinatorics of very large spaces of explanatory hypotheses. Structured conclusions can be arrived at whose parts are connected by relationships of type-subtype, statistical association, and explainer-explained. An instance of this machine exists, exercising some of the capabilities we attribute to the abstract machine, and arriving at correct answers in complicated situations.

Although the mechanism is an abstraction of the architecture of the Red-2 system, DENDRAL, INTERNIST, and systems based on the Set-Covering model of abduction realize it too, in varying degrees. In this light the present offering should be seen, not so much as contributing new mechanisms, but as showing how existing systems can be analyzed; and how, once we understand the tasks, efficient mechanisms can be devised specifically to achieve them.

More grandly we may say that a computational description has been given to the functional architecture of a possible mind, or rather, of a certain dimension or slice of a possible mind. The kind of synthesis of explanatory hypotheses we describe here is a *generic task of higher intelligence*. It must be accomplished somehow by any intelligent, knowledge-using agent that comes to "know" by calling upon "concepts", attaching them to situations or objects, and using the resulting little hypotheses as materials to form composite "best explanations". The task is general, but specific. There are a limited number of functional architectures that could accomplish it, especially when account must be taken of the constraints imposed by limited knowledge, limited time, and limited computational resources. There are even fewer architectures that are *especially suited* to the task, and we have just described one of them.

7. Acknowledgments

This work has been supported in various stages by NSF Grant MCS-8305032, NIH Grant R01 LM 04298 from the National Library of Medicine, and DARPA/RADC contract F30602-85-C-0010. Dr. Jack W. Smith, Jr. is supported by NLM Career Development Award K04 LM00083. Thanks are due to Tom Bylander for showing that the task of producing a consistent composite is NP complete, and for his helpful comments on a previous draft. Also to Bill Punch and Dean Allemang for their discussions and encouragement of the approach to abduction, to Jon Sticklen for arguing with the ideas until they were better justified, to the members of the recent graduate seminar at Ohio State on diagnostic reasoning for their helpful comments, and to two anonymous reviewers of a previous draft of the paper who, by their

failure to understand what was being presented, pointed the way to an improved explication of the ideas.

References

1. Marr, David, *Artificial Intelligence: A Personal View*, The MIT Press, 1981, Also appears in Artificial Intelligence 9(1):47-48, 1977.
2. Marr, David, *Representing and Computing Visual Information*, The MIT Press, 1979.
3. B. Chandrasekaran and J. Josephson, "Explanation, Problem Solving, and New Generation Tools: A Progress Report", *Proceedings of the Expert Systems Workshop*, April 1986, pp. 122-126.
4. B. Chandrasekaran, "Towards a Taxonomy of Problem-Solving Types", *AI Magazine*, Vol. Winter/Spring, 1983, pp. 9-17.
5. T. Bylander and S. Mittal, "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling", *AI Magazine*, Vol. 7, No. 3, August 1986, pp. 66-77.
6. F. Gomez and B. Chandrasekaran, *Knowledge Organization and Distribution for Medical Diagnosis*, Addison-Wesley Publishing Company, 1984, pp. 320-338, Also appears in IEEE Transactions on Systems, Man and Cybernetics, SMC-11(1):34-42, January, 1981
7. David C. Brown, "Expert Systems for Design Problem-Solving Using Design Refinement with Plan Selection and Redesign". Ph.D. Dissertation, The Ohio State University
8. D. C. Brown and B. Chandrasekaran, "Plan Selection in Design Problem-Solving", *Proceedings of AISB85 Conference*, The Society for AI and the Simulation of Behavior (AISB), Warwick, England, April 1985.
9. D. Herman, J. Josephson, and R. Hartung, "Use of DSPL for the Design of a Mission Planning Assistant", *Expert Systems in Government Symposium*, October 1986, pp. 273-278.
10. S. Mittal, B. Chandrasekaran and J. Sticklen, "PATREC: A Knowledge-Directed Data Base for a Diagnostic Expert System", *IEEE Computer Special Issue*, Vol. 17, September 1984, pp. 51-58.
11. Ronnie Sarkar, "State Abstraction Problem Solving". OSU LAIR Technical Report
12. W. F. Punch III, M. C. Tanner and J. R. Josephson, "Design Considerations for PIERCE, a High-Level Language for Hypothesis Assembly", *Expert Systems in Government Symposium*, October 1986, pp. 279-281.
13. Josephson, J. R.; Chandrasekaran, B.; Smith, J.W., "Assembling the Best Explanation", *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, IEEE Computer Society, Denver, Colorado, December 3-4 1984, pp. 185-190, A revised version by the same title is now available as a technical report.
14. J. R. Josephson, M. C. Tanner, J. W. Smith, M.D., J. Svirbely and P. Straum, "Red: Integrating Generic Tasks to Identify Red-Cell Antibodies", *Proceedings of The Expert Systems in Government Symposium*, Kamal N. Karna, ed., IEEE Computer Society Press, 1985, pp. 524-531.
15. Smith, Jack W.; Svirbely, John R.; Evans, Charles A.; Strohm, Pat; Josephson, John R.; Tanner, Mike, "RED: A Red-Cell Antibody Identification Expert Module", *Journal of Medical Systems*, Vol. 9, No. 3, 1985, pp. 121-138.
16. Smith, J., Josephson, J., Tanner, M., Svirbely J., Strohm, P., "Problem Solving in Red Cell Antibody Identification: Red's Performance on 20 Cases", Tech. report, Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, 1986.
17. Doyle, Sir Arthur Conan, *The Sign of the Four*, Clarkson N. Potter, Inc., 1960.
18. Peirce, C.S., *Abduction and Induction*, Dover, 1955, pp. 150ff., ch. 11.
19. Gilbert Harman, "The Inference to the Best Explanation", *Philosophical Review*, Vol. LXXIV,

January 1965, pp. 88-95.

20. Ennis, R., "Enumerative Induction and Best Explanation", *The Journal of Philosophy*, Vol. LXV, No. 18, September 1968, pp. 523-529.
21. John R. Josephson, *Explanation and Induction*, PhD dissertation, The Ohio State University, 1982.
22. William G. Lycan, "Epistemic Value", *Synthese*, Vol. 64, 1985, pp. 137-164.
23. John McCarthy, "Circumscription - A Form of Non-Monotonic Reasoning", *Artificial Intelligence*, Vol. 13, 1980, pp. 27-39.
24. John R. Josephson, *Explanation and Induction*, PhD dissertation, The Ohio State University, 1982.
25. John R. Josephson, *Explanation and Induction*, PhD dissertation, The Ohio State University, 1982, pages 77ff
26. Eugene Charniak and Drew McDermott, *Introduction to Artificial Intelligence*, Addison Wesley, 1985.
27. Pople, H., "On the Mechanization of Abductive Logic", *Proceedings of the Third International Joint Conference on Artificial Intelligence*, 1973, pp. 147-152.
28. Reggia, J., "Abductive Inference", *Proceedings of The Expert Systems in Government Symposium*, Kamal N. Karna, ed., IEEE Computer Society Press, 1985, pp. 484-489.
29. Thomas A. Sebeok and Jean Umiker-Sebeok, "You Know My Method": *A Juxtaposition of Charles S. Peirce and Sherlock Holmes*, Indiana University Press, Bloomington, 1983, ch. 2.
30. Marcello Truzzi, *Sherlock Holmes: Applied Social Psychologist*, Indiana University Press, Bloomington, 1983, ch. 2.
31. Miller, R.A., Pople, J.E. Jr. and Myers, J.D., "INTERNIST - I, An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine", *New England Journal of Medicine*, Vol. 307, 1982, pp. 468-476.
32. Buchanan, B. G., Sutherland, G. L. and Feigenbaum, E. A., *Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry*, Edinburgh University Press, Edinburgh, 1969.
33. Reggia, J., "Diagnostic Expert Systems Based on a Set Covering Model", *International Journal of Man-Machine Studies*, Vol. 19, November 1983, pp. 437-460.
34. James A. Reggia, Barry T. Perricone, Dana S. Nau, and Yun Peng, "Answer Justification in Diagnostic Expert Systems--Part 1: Abductive Inference and Its Justification", *IEEE Transactions on Biomedical Engineering*, Vol. BME-32, No. 4, April 1985, pp. 263-267.
35. M. Garey and D. Johnson, *Computers and Intractability*, W. H. Freeman and Company, New York, 1979.
36. Johan de Kleer, "Reasoning About Multiple Faults", *AI Magazine*, Vol. 7, No. 3, August 1986, pp. 132-139.
37. V. Sengugamoorthy and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems", *Experience, Memory, and Reasoning*, 1986, pp. 47-73.
38. Genesereth, M.R., "The use of design descriptions in automated diagnosis", *Artificial Intelligence*, Vol. 24, December 1984, pp. 411-436, PA
39. B. Chandrasekaran, F. Gomez, S. Mittal, and J. W. Smith, "An Approach to Medical Diagnosis Based on Conceptual Structures", *Proceedings of the Sixth International Joint Conference on*

Artificial Intelligence, 1979, pp. 134-142.

40. B. Chandrasekaran and S. Mittal, "Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems", in *Advances in Computers*, M. Yovits, ed., Academic Press, Vol. 22, 1983, pp. 217-293.
41. Pople, H., "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI 77, 1977, pp. 1030-1037.
42. Newell, A. and Simon, H.A., *GPS. A Program That Simulates Human Thought*, McGraw-Hill, New York, 1963, pp. 279-293.
43. D. Allemang, M. C. Tanner, T. Bylander and J. R. Josephson, "On the Computational Complexity of Hypothesis Assembly", *Proc. Tenth International Joint Conference on Artificial Intelligence*, Milan, August 1987, to appear

Appendix G

Design: An Information Processing-Level Analysis

**The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research**

**Technical Report
August 1987/Revised January 1988**

Design: An Information Processing-Level Analysis

**B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210**

Note: This is a draft version of Chapter 2 of the book *Design Problem Solving: Knowledge Structures and Control Strategies* by D. C. Brown and B. Chandrasekaran (forthcoming).

ABSTRACT

Design problem solving is analyzed as an information-processing task: the task and its information requirements are analyzed. This analysis suggests possible decompositions of the task into a number of subtasks, depending upon what kinds of knowledge are in fact available in a domain. This decomposition can be carried on several levels until we reach an understanding of how various generic problem solving capabilities can come together to help solve the design problem. This analysis suggests possible problem solving architectures for design. A number of AI approaches to design are discussed in this perspective and it is shown how each of them can be understood as solving a particular version of the design problem, using one of the architectures that arises from the analysis in such a way that the architecture matches the knowledge available in the domain.

Table of Contents

2. Design: An Information Processing-Level Analysis	1
2.1. A Framework for Design Problem-Solving	1
2.1.1. What is the Design Problem?	1
2.2. What Kind of Space in Which To Search?	2
2.3. Information processing Analysis of Design	3
2.3.1. Processes That Propose Design Choices	3
2.3.2. Auxiliary Processes	7
2.4. Implications of Above Analysis	8
2.5. Classes of Design	9
2.5.1. Class 1 Design	9
2.5.2. Class 2 Design	9
2.5.3. Class 3 Design	9
2.5.4. A Class 3 Product	10
2.5.5. Class 3 Complexity	10

2. Design: An Information Processing-Level Analysis

2.1. A Framework for Design Problem-Solving

2.1.1. What is the Design Problem?

In this chapter we look at design as an information processing task: i.e., specify what kinds of input and output characterize the task abstractly. This can then form the basis for investigating what kinds of knowledge and inference processes can help solve what parts of the task. We will avoid talking in terms of particular methods of representation of knowledge, say, rules or frames, but concentrate instead on *what* needs to be represented, and what types of inferences are needed. Once the nature of the subtasks in design becomes clear, then the question of how to implement them can be undertaken. The reader might remember that one of our criticisms of the expert system area has been that implementation level phenomena have been allowed to interfere with an analysis of task-level phenomena. We would like to keep them clearly separate.

Design is a very complex activity and covers a wide variety of phenomena: from planning a day's errands to theory construction in science to composing a fugue are design activities. In order to give some focus and use some shared intuitions, let us restrict the scope in this discussion to the design of artifacts that satisfy some goals.

A designer is charged with specifying how to make an artifact which satisfies or delivers some goals. For each design task, the availability of a (possibly large and generally only implicitly specified) set of *primitive components* can be assumed. The domain also specifies a repertoire of *primitive relations* or connections possible between components. An electronics engineer, e.g., may assume the availability of transistors, capacitors, etc., of various types when he is designing a waveform generator, and examples of primitive relations in that domain are serial and parallel connections between components. Similarly, an architect might assume the availability of building materials. If the architect has to design an unusual brick as part of his architectural specification, at least he can assume the availability of clay, and so on. Of course design can also be recursive: if a certain component that was assumed to be available is in fact not available, the design of that can be undertaken at the next round, and the domain for the component may be rather different than the original domain, as in the example of building and brick design. If the component design is not successful, the original design may be discarded and the task undertaken again.

The design task can then be specified as follows:

- *Complete specification of a set of 'primitive' components and their relations so as to meet a set of constraints.*

Some of the constraints will refer to the functions or goals of the artifact, some may pertain to the parameters of the artifact (e.g., 'total weight to be less than 1 ton'), yet others may provide constraints on the design process itself, and finally constraints may apply to the process of assembling the artifact (manufacturing constraints). Often the goals may not be stated explicitly or in sufficient detail at the start of the design process. In hard design problems, the world of primitive objects may be very open-ended. In spite of all such caveats, the above working definition is a good starting point for our discussion.

This definition also captures the *domain-independent* character of design as a generic activity at some level of abstraction. Planning, programming and mechanical design all share the above definition to a significant degree. Note that the knowledge needed and many of the detailed mechanisms will of course be domain-specific. For example, mechanical design may call for significant amounts of spatial reasoning, while the electrical domain may only involve topological reasoning. But the nature of the design problem as a whole has many commonalities at the level of the above definition, and as we shall see, at the level of many of the subprocesses.

This definition is not meant to imply the existence of one method for all design. The main message of our work is that design actually consists of a large number of distinct processes that work together, each contributing some information needed during the design process. In fact the apparent difference in the

design process in different domains and different designers can be explained by the dominance of some of these subprocesses over the others due to differences in the knowledge available.

The above definition gives a clue to why the design problem can be *hard* for AI and often also for people. In realistic domains the size of the set of primitive objects is quite large and implicit. The design problem is formally a search problem in a very large space for objects that satisfy multiple constraints. Only a vanishingly small number of objects in this space constitute even "satisficing", not to speak of optimal, solutions. What is needed to make design practical are powerful strategies that radically shrink the search space.

2.2. What Kind of Space in Which To Search?

The idea of search in a state space goes back to the early days of AI, and Newell [Newell, 1980] has formalized the *Problem Space Hypothesis* essentially stating that all goal-directed behavior takes place in some problem space.

Before search can take place, the problem space needs to be defined. But design problem solving does not have a unique problem space. Different kinds of problem spaces can be visualized, each appropriate for some kinds of domain knowledge and not others. For search in a problem space to be operationally definable, problem states, operators which transform one problem state into a set of successors, and some ordering knowledge that helps to choose between alternatives need to be available. For search to be practical, generation of successors and choice among alternatives should not themselves be complex problem solving activities. The last condition means that domain knowledge should be directly available which can be applied to generate successors and choose among them.

Let us consider the *transformation* approach to design [Barstow, 1984; Balzer, 1981] as a concrete illustration of these issues. We can consider the set of specifications to be the initial state, and a fully designed artifact to be the end state. Operators transform parts of the specifications into alternative design commitments that will realize them. So an intermediate state will consist of design commitments which realize some of the specifications along with remaining specifications. The process of design can be thought of as searching for a series of design commitments that results in a goal state.

While this is formally satisfactory, knowledge may not be available in all domains for successor generation and alternative selection. In the programming domain to which this idea has been applied, there seem to be several examples where knowledge of this form is in fact available. However, this problem space is not of general applicability. (No single problem space is.) In some domains, the constraints as stated may not be factorizable in this way, and there may be significant interactions between the designs that are chosen to meet parts of the constraints. There is also no guarantee that the design process can always correspond to incremental choices. Large subsystems may be designed first and then only design within subsystems may proceed. Thus the actual design process in that domain may not correspond to navigation in this transformation problem space. Knowledge may be directly available which cuts a swath across the space, so that several constraints together are realized by a precompiled design that is recognized as applicable (design plans). Finally, in many domains, the problem is reformulated by a decomposition so that a number of disjoint local spaces, each corresponding to a subproblem are created. (We will discuss decomposition and design plans shortly in greater detail.)

The point of the discussion is this. Which problem space is used depends on the forms in which domain knowledge for representation and control are available. Using an inappropriate problem space will result in artificial heuristic functions being used which do not capture the real structure of domain knowledge.

We propose that in design problem solving a *variety of types of knowledge* can be identified, each of which helps solve a portion of the design problem in a computationally efficient way. Expertise consists of an accumulation of a repertoire of such knowledge. However, unlike the current view in the expert systems area, this expertise is not viewed as collections of pieces of knowledge, to be used by a uniform inference technique. Instead, knowledge comes in various *generic forms*, each structured in

characteristic ways and using inference methods that are appropriate to it. Each type of knowledge can produce some information that may be needed or useful during design, or can generate a part of the design solution. Conversely, each type of knowledge *requires* information of certain types to be available before it can be useful.

Thus the picture that we would like to give of design problem solving is as a cooperative activity between multiple types of problem solvers, each solving a subproblem using knowledge and inference of specific types, and communicating with other computational modules or problem solvers for information that is needed for it to perform its task, or to deliver information that they need for their tasks. Thus an analysis of design as problem solving consists of identifying these subprocesses, their information processing responsibilities, and the knowledge and inference needed to deliver these functions. We call this kind of analysis an *information processing analysis* of design. This is the task of the next section.

2.3. Information processing Analysis of Design

The style of analysis will be to identify subtasks in design, and characterize what kinds of information or solution they are responsible for providing. Some of these subtasks can be performed in a number of different ways: an AI solution is only one way. For example, during design, it will be necessary to find if a certain design requirement is met. A traditional computational algorithm may be able to do that in some cases, e.g., finding out if stress in a member is less than a certain amount may be done by invoking a finite element analysis algorithm. Sometimes this information may require an AI-type solution, involving an exploration of some space in a qualitative way, e.g., by doing a qualitative simulation of the artifact. In what follows we will only describe issues associated with AI-type solutions for these subtasks, but the larger possibility needs to be kept in mind in the actual design of knowledge-based systems for design.

During the discussion we will try to relate the framework to a number of previous and current approaches to design. But the literature on design is vast. Even within AI, work on design has proliferated over the last decade. We do not intend to be exhaustive in our coverage. Our intent is to point to some of the other work as a way to illuminate the discussion.

We will describe a number of subprocesses or subtasks in design and describe the role they play in design. The design process can be usefully separated into those processes that play a role in the "generate" part and those that help in the "test" part. We subdivide our discussion into two groups of processes: those that are responsible for proposing or making design commitments of some sort, and those that serve an "auxiliary" role, i.e., generate information needed for the proposers, and help test the proposed design.

2.3.1. Processes That Propose Design Choices

(1): **Decomposition.** This is a very common subpart of the design activity. We will use this process as an example of information processing analysis, and describe it in terms of all the features that such an analysis calls for: types of knowledge, information needed, and the inference processes that operate on this form of knowledge.

Knowledge of the form $D \rightarrow \{D_1, D_2, \dots, D_n\}$, where D is a given design problem, and D_i 's are "smaller" subproblems (i.e., associated with smaller search spaces than D) is often available in many domains. In many domains, there may be a number of alternate decompositions available, and so choices (and possible backtracking) will need to be made in the space of possible decompositions. Repeated applications of the decomposition knowledge produce *design hierarchies*. In well-trodden domains, effective decompositions are known and little search at that level needs to be conducted as part of routine design activity. For example, in automobile design, the overall decomposition has remained largely invariant over several decades.

Dependable decomposition knowledge is extremely effective in controlling search since the total search space is now significantly smaller. This power arises from the fact that decompositions represent a previously compiled solution to a part of the design problem, and thus at run-time the design problem solver can avoid this search.

Information Needed: The decomposition process needs two kinds of additional information for it to be effective.

- How the goals or constraints on D get translated into constraints on the subproblems D1,... Dn.
- How to glue the designs for D1, D2,... Dn into a design for D.

Information of the above types may be given as part of the decomposition knowledge or can be obtained by accessing another processor which can produce that information. We will shortly refer to a method called *constraint posting* that has been proposed for generating constraints on subproblems. How to glue the designs for subproblems may require additional problem solving, such as simulating D1 and D2, e.g., and finding out exactly where and how the gluing can occur. The Critter system [Kelly, 1984], e.g., provides such a simulation facility applicable under certain assumptions that helps both in generating constraints for the subproblems and in gluing the solutions together.

Inference Process: There are two sets of inference processes, one dealing with which sets of decompositions to choose, and the other concerned with the order in which the subproblems within a given decomposition ought to be attacked. (Remember that a decomposition merely converts a design problem into a set of presumably "smaller" problems, which still need to be solved for the decomposition to be successful.)

For the first problem, in the general case, the decomposition will produce an AND/OR node, i.e., will produce decompositions some of which are alternatives and others all of which need to be solved. Finding the appropriate decomposition may involve searching in a space represented as an AND/OR graph. But as a rule such searches are expensive. Routine design problems should not require extensive searches in the decomposition space. To avoid the search problem but to use domain knowledge about decomposition, human-machine interaction between human experts and machine processing can be arranged so that the machine proposed alternative decompositions, and the human chooses the most plausible ones. Precisely this sort of shared labor is used in the VEXED system [Mitchell, et al, 1985] during its problem decomposition phase.

The problem of the order in which to attack the problems in the decomposition list when combined with the problem of searching in the decomposition space can make the total search very complex, since the investigation of the subproblems in a given decomposition will be in general non-reusable if that decomposition turns out not to be successful. This explains the extreme difficulty of the design problem in the general case. However, in most cases of routine design, the decomposition knowledge leads to a design hierarchy as mentioned. The default control process for investigating within a given design hierarchy is then *top-down*. While the control is top down, the actual sequence in which design problems are solved may occur in any combination of top-down and bottom up manner. For example, in designing an electronic device, a component at the tip level of the design hierarchy may be the most limiting component and many other components and subsystems can only be designed after that is chosen. The actual design process in this case will appear to have a strong bottom up flavor. Control first shifts to the bottom-level component, and the constraints that this component design places on the design of other components are passed up.

A related issue is one of whether the control should be depth-first or breadth first. Again, this is very much a function of the domain. The specification language for control behavior in this process should be expressive enough for a variety of control possibilities along these lines.

Decomposition is an ubiquitous strategy in AI work in design. McDermott's NASL system [D.V.McDermott, 1978] uses this extensively. Freeman and Newell [Freeman and Newell, 1971] discuss various decomposition criteria, including functional and structural. The transformational design work of Barstow and Mostow uses decomposition in a degenerate form: the constraint set is such that subsets of it correspond to different design problems, and so can be separately expanded.

(2): **Design Plans.** Another pervasive form of design knowledge, again that represents precompiled partial design solutions, is a *design plan*. A design plan specifies a sequence of design actions to take for

producing a piece of abstract or concrete design. In abstract design, choices are made which need to be further "expanded" into concrete details at the level of primitive objects. These design plans are indexed in a number of ways, two of them being indexing by design goals (*for achieving <goals>, use <plan>*), or by components (*for designing <part>, use <plan>*). Since plans may have steps that point to other plans, design plans can subsume decomposition knowledge. From the viewpoint of complexity reduction, the central contribution plans make is as an encoding of previous successful exploration of a problem space by abstracting from the experience of an individual expert or a design community in solving particular design problems.

Each goal or a component may have a small number of alternate plans attached to them, with perhaps some additional knowledge that helps in choosing among them. A number of control issues arise about abandoning a plan and backing up appropriately, or *modifying* a plan when a failure is encountered.

The inference process that is applicable can be characterized as *instantiate and expand*. That is, the plan's steps specify some of the design parameters, and also specify calls to other design plans. Choosing an abstract plan and making commitments that are specific to the problem at hand is the instantiation process, and calling other plans for specifying details to portions is the expansion part.

A number of additional pieces of information may be needed or generated as this expansion process is undertaken. Information about dependencies between parts of the plan may need to be produced at runtime (e.g., discovering that certain parameters of a piston would need to be chosen before that of the rod), and some optimizations may be discovered at run time (e.g., the same base that was used to attach component A can also be used to attach component B). For example, Noah [Sacerdoti, 1977] can be understood as a system that instantiates and expands design plans. In Noah, corresponding to each goal of the artifact under design, there is a stored procedure which can be interpreted as a design plan. These plans can call other procedures/plans until a hierarchy is created. Noah concentrates its problem solving on recognizing ordering relations and redundancies between the components of the plan.

The idea of design plans has been used successfully in the domain of programming or algorithm design [Rich, et al, 1979], [Johnson and Soloway]. The notion that *plans* constitute a very basic knowledge structure has been with us from the 1950's when this idea was discussed extensively by Miller, Galanter and Pribram [Miller, et al, 1960]. Schank and Abelson [Schank and Abelson, 1977] have also discussed the use of plans as a basic unit of knowledge. The Molgen work of Friedland [Friedland, 1979] uses design plans as a basic construct. More recently Mittal's PRIDE system [Mittal, et al, 1986] has used them for design knowledge representation.

(3): Design by Critiquing and Modifying Almost Correct Designs. A variation on the design plan idea is that the designer has a storehouse of actual successful designs indexed by the goals and constraints that they were designed to satisfy. Sussman [Sussman, 1973] has proposed that a design strategy is to choose an already-completed design that satisfies constraints closest to the ones that apply to the current problem, and modify this design for the current constraints. This process needs information of the following kinds.

- **Matching:** How to choose the design that is "closest" to the current problem? Some notion of prioritizing over goals or differences in the sense of *means-ends analysis* may be needed, if this information cannot be generated by a compiled matching structure. In some cases, some analogical reasoning capabilities may be appropriate by which to recognize "similar" problems.
- **Critiquing:** Why does the retrieved design fail to be a solution to the current problem? This analysis is at the heart of learning from failure, and sophisticated problem solving may be needed to analyze the failure. This capability of critiquing a design is of more general applicability than for this particular design process.
- **Modifying:** How to modify the design so as to meet with the current goals? In many cases, this information may be available in a compile form, but in general, this also requires sophisticated problem solving.

The processes of critiquing and modifying have more general applicability than as parts of this particular design process. We discuss criticism as one of the auxiliary processes later in this section. Design modification, however, is a useful process in the "generate" part of design, so we discuss some of the issues related to it here.

Modification as a subprocess takes as input information about failure of a candidate design and changes the design. Depending upon the sophistication about failure analysis and other forms of knowledge available, a number of problem solving processes are applicable:

- A form of means-ends reasoning, where the differences are "reduced" in order of most to least significant.
- A kind of hill-climbing method of design modification, where parameters are changed, direction of improvement noted, and additional changes are made in the direction of maximal increment in some measure of overall performance. This form can even constitute the only method of design in some domains: assign arbitrary values to the parameters, and change them in a hill-climbing fashion until a maximum is reached, and deliver that as the design. This is especially applicable where the design problem is viewed as a parameter choice problem for a predetermined structure. The system called DOMINIC [Howe, et al] engages in this form of design problem solving.
- Dependencies can be explicitly kept track of, in such a way that when a failure occurs, the dependency structure directly points to where a change ought to be made. *Dependency-directed backtracking* was proposed by Stallman and Sussman [1977] as one approach to this problem. Mittal [Mittal and Araya, 1986] proposes a variation on dependency tracking for modification of designs on failure.
- What to do under different kinds of failures may be available as explicit domain knowledge in routine design problems. This information can be attached to the design plans. The work to be described in later chapters uses this highly compiled form.

Sussman [1973] has investigated the retrieval and use of previous designs in circuit design. Schank [1983] has been an advocate of case-based reasoning for a variety of problems.

(4): **Design by Constraint Processes.** For some design problems a process of simultaneous constraint satisfaction by constraint propagation can be employed. In order for this to work computationally effectively, it is best if the structure of the artifact is known and design consists of selecting parameters for the components. Constraints can be propagated in such a way that the component parameters are chosen to incrementally converge on a set that satisfies all the constraints. Macworth [1977] provides a good discussion of several techniques for this. This is an instance of what is called, in optimization theories, *relaxation procedures*¹. Human problem solvers aren't particularly good at this form of information processing without pencil and paper. The incremental convergence process can be treated as a form of problem space exploration, so we are including it in this discussion.

Constraint satisfaction processes can be viewed as applying design modification repeatedly and incrementally. Thus many of comments we made earlier regarding design modification are applicable. In particular, some of the constraint propagation techniques can be viewed as versions of *hill-climbing* methods in search. And variations such as dependency-directed changes to parameters can be adopted during each modification cycle. More complex processes such as *constraint-posting* can be used where additional constraints are generated as a result of choices made for earlier parameter choices. These constraints are used for remaining parameter choices.

Configuration problems are an interesting and well-known class of problems (made famous by the R1 system [J. McDermott, 1982]) in design. Some versions of them can be decomposed into subproblems

¹Unfortunately, this use of the term *relaxation* interferes with another use of it in design, viz., *relaxing the constraints* so that a hard design problem may be converted into a relatively easier one.

whose solutions can be neatly glued back together. In fact, R1's problem solving is done as a linear series of subtasks. However, in the general case, these problems often have no clear decomposition into subproblems, because of extensive interactions between various parts of the design. On the other hand, many configuration problems have the tractable feature that most of the components of the device are already fixed, and only their connections and a few additional components to mediate the connections need to be chosen. This makes iterative techniques applicable by making it likely that one can converge on the solution. Constraint satisfaction methods are often applicable to configuration problems. Marcus, McDermott and Wang [1985] discuss a strategy called *propose and revise*, where commitments are made for some parts of the design, which generates additional constraints, and if later parts in the design problem cannot be solved, earlier commitments are revised. Frayman and Mittal [1987] discuss the configuration task abstractly.

Caution! Formally all design can be thought of as constraint satisfaction, and one might be tempted to propose global constraint satisfaction as a universal solution for design. The problem is that these methods still can constitute a fairly expensive way to search the space. For example, *propose and revise* can end up searching the entire space in difficult problem spaces and hill-climbing methods can get stuck at local optima. Hence these methods are not a universally applicable for practical design. Other methods of complexity reduction such as problem decomposition are still very important in the general case. They can create subproblems with sufficiently small problem spaces in which constraint satisfaction methods can work without excessive search.

Human problem solvers need computational assistance in executing constraint satisfaction approaches: the methods are computationally intensive and place quite a burden on short term memory. As long as attempts are not made to use them as universal design methods, they can be effective computational techniques for portions of the design problem.

2.3.2. Auxiliary Processes

So far the subprocesses in design that we have considered:

- decomposition, design plan instantiation and expansion, modification of an almost correct design, constraint satisfaction,

contribute to design by proposing some design commitments. Along the way, we have referred to some other processes which serve the former by providing information that they need. Let us discuss them briefly here.

(1): **Goal/Constraint Generation for Subproblems.** Given a decomposition $D \rightarrow \{D_1, D_2, \dots, D_n\}$, one will need to know how the goals/constraints of D are translated into goals/constraints for the subproblems. In many domains, this information is part of the decomposition knowledge. But if it is not available, additional problem solving is called for. The literature on constraint-posting that we referred to earlier proposes methods applicable in some cases.

Vexed [Sternberg] provides an example of constraint generation for subproblems given a particular problem decomposition. In this domain the subproblems have a serial connection relation. For example, D may be implemented by two modules D_1 and D_2 connected in series. A constraint propagation scheme (called CRITTER [Kelly, 1984]) takes the input to D and produces the constraints on D_1 's output/ D_2 's input. Design of D_1 and D_2 can then proceed.

(2): **Recomposition.** We alluded to this in our discussion on decomposition: how to glue the solutions of the subproblems back into a solution for the original problem. Integrating them may require simulating the subdesigns and find how they interact. Or other methods of problem solving may be called for. Scientific theory building involves assembling portions of theories into larger coherent theories, and needs powerful interaction analyses. RED [Josephson, et al, 1987] proposes an especially powerful strategy for composing explanatory theories.

(3): **Design Verification:** This is part of the "test" component of the design activity: whether a candidate design delivers the functions and meets with any other relevant constraints. In most cases, it can be done by straightforward compiled computational methods, e.g., "add weights of components and

check that it is less than x ," or invoking possibly complex mathematical formulae, such as a finite element analysis, that does not involve problem solving. In some cases, additional problem solving may be called for for verification. For instance, qualitative simulation of a piece of machinery to decide if any of its parts will be in the path of another part may be needed for verifying a proposed design.

(4): **Design Criticism.** At any stage in design, any failure calls for analyzing the candidate design for reasons for failure. This form of criticism played a major role in the method of design by retrieving an almost correct design. In most routine design, fairly straightforward methods will suffice for criticism, but in general this calls for potentially complex problem solving. *Design modification* uses the results of criticism.

2.4. Implications of Above Analysis

The analysis of the design process in terms of subprocesses with well-defined information processing responsibilities helped us in identifying types of knowledge and inference needed. This in turn directly suggests a *functional architecture* for design with these subprocesses as *building blocks*. It also suggests a principled way in which to define the human-machine interaction in design. Firstly, whenever knowledge and control can be explicitly stated for one of the modules or building blocks, that module can be built directly, by using a knowledge and control representation that is appropriate to that task. Secondly, if knowledge for a module is not explicitly available, the human can be part of the loop for providing information that that module would have been responsible for. For example, failure analysis and common sense reasoning involving space and time are difficult problem solving tasks. These tasks may be needed for the performance of design modification and design verification, respectively. The human/machine division of responsibility may be done in such a way that the machine turns to the user for the performance of these tasks. As these tasks are better understood, they can be incrementally brought into the machine side of the human/machine division of labor.

Another kind of human machine interaction is possible in this framework. Note that each subprocess is characterized both by specific types of knowledge and by inference and related control problems. We mentioned, e.g., that search in the space of problem decompositions can become quite complex. One way in which a module can interact with a domain expert is by proposing available knowledge and letting the human make the control choices by using knowledge that has not been made explicit in the problem solving theory. As a practical matter, this can be an effective way of using the module as a knowledge source, even without a complete theory of problem solving using that knowledge. The VEXED system that we have mentioned in fact works in this mode: it proposes possible decompositions, and the user is asked to choose the one he or she would like to pursue. Similarly, when a design system's choice of design plans fails, it may turn to the user for choosing alternative plans.

Let us elaborate on the functional architecture for design that results from this analysis. Because each subprocess uses characteristic types of knowledge and inference, a "mini-shell" can be associated with it and knowledge and inference can be directly encoded using that shell. Since each of the tasks has a clear information processing responsibility, the modules can communicate with each other in terms of the information that defines the input and outputs of these modules. Thus the modularity that results is a task-level modularity.

In the rest of this book, we provide the details of the functional architecture for one type of design, a form of *routine design* that we have termed Class 3 Design. This way of analyzing design and identifying architectures out which design problem solvers can be built is what is novel about the point of view of the book. We will soon proceed to a description of our informal classification of design problems, but before that we need to take note of some other suggestions that have been made for design problem solving and relate them to our analysis.

2.5. Classes of Design

The above analysis of design subprocesses can be used to provide an *informal* classification of design problems. Many of the processes in the "test" part of design, such as design verification by qualitative simulation, can be arbitrarily complex, but they are not particularly specific to design. The design process simply calls upon these other problem solving skills. On the other hand, many of the processes in the "generate" portion are quite specific to design as a problem solving process, so our classification is based largely on the subprocesses in the "generate" part of design.

Each of the processes

- decomposition, design plan instantiation and expansion, modification of similar designs, constraint satisfaction,

performs some aspect of design, using information either directly available or supplied by auxiliary problem solving or other computational processes. Each of them comes with a set of control problems that can be more or less complex, and needs knowledge in certain forms.

The framework suggests that design by decomposition, i.e., breaking problems into subproblems, by plan synthesis where necessary, and by plan selection where possible, are the core processes in knowledge-based design, i.e., it gives importance to the first two processes in the above list as the major engines of complexity reduction in design. The classification is largely based on the difficulty of these subtasks or processes, in particular on the completeness of knowledge, the ready availability of the needed auxiliary information and the difficulty of the control issues.

2.5.1. Class 1 Design

This is open-ended "creative" design. Goals are ill-specified, and there is no storehouse of effective decompositions, not to speak of design plans for subproblems. Even when decomposition knowledge is available, most of the effort is in searching for potentially useful problem decompositions. For each potential subproblem, further work has to be done in evaluating if a design plan can be constructed. Since the design problem is not routine, considerable problem solving for many of the auxiliary processes will need to be performed.

The average designer in industry will rarely, if ever, do Class 1 design, as we consider this to lead to a major invention or completely new products. It will often lead to the formation of a new company, division, or major marketing effort. This then is extremely innovative behavior, and we suspect that very little design activity is in this class.

2.5.2. Class 2 Design

Class 2 design is characterized by powerful problem decompositions already available, but design plans for some of the component problems in need of *de novo* construction or substantial modification. Design of a new automobile, e.g., does not involve new discoveries about decomposition: the structure of the automobile has been fixed for quite a long time. On the other hand, several of the components in it constantly undergo major technological changes, and routine methods of design for some of them may no longer be applicable.

Complexity of failure analysis will also take a problem away from routine design. Even if design plans are available, if the problem solver has to engage in very complex problem solving procedures in order to decide how to backtrack, the advantage of routine design is reduced. In short, whenever substantial modifications of design plans for components are called for, or when synthesis in the design plan space is especially complicated, we have a Class 2 problem.

2.5.3. Class 3 Design

This is relatively routine design: effective problem decompositions are known, compiled design plans for the component problems are known, and actions to take on failure of design solutions are also explicitly known. There is very little complex auxiliary problem solving needed. In spite of all this simplicity, the design task itself is not trivial: complex backtracking can still take place. The design task is

still too complex for simple algorithmic solutions or table look up.

Class 3 problems are routine design problems, but still requiring knowledge-based problem solving. The ensuing chapters of this book deal with an approach to building knowledge-based systems for routine design problems of this type. The processes described here can work in conjunction with auxiliary problem solvers of various types, but the theory for them is not developed further in this book. The examples used all assume that the information to be provided by the auxiliary design processes, e.g., design criticism, verification, and subproblem constraint generation, are all available in a compiled manner.

2.5.4. A Class 3 Product

In a large number of industries, products are tailored to the installation site while retaining the same structure and general properties. For example, an Air-cylinder intended for accurate and reliable backward and forward movement of some component will need to be redesigned for every new customer in order to take into account the particular space into which it must fit or the intended operating temperatures and pressures. This is a design task, but a relatively unrewarding one, as the designer knows at each stage of the design what the options are and in which order to select them. Note that that doesn't mean that the designer knows the complete sequence of steps in time (i.e., the trace) in advance, as the designer has to be in the problem-solving situation before each decision can be made. There are just too many combinations of requirements and design situations to allow an algorithm to be written to do the job.

As this tends to be unrewarding work for humans and as this type of non-trivial problem appears to be possible to do by computer there is strong economic justification for us to attack this problem.

2.5.5. Class 3 Complexity

The complexity of the class 3 design task is due not only to the variety of combinations of requirements, but also to the numerous components and sub-components, each of which must be specified to satisfy the initial requirements, their immediate consequences, the consequences of other design decisions, and the constraints of various kinds that a component of this kind will have.

While class 3 design can be complex overall, at each stage the design alternatives are not as open-ended as they might be for class 2 or 1, thus requiring no planning during the design. In addition, all of the design goals and requirements are fully specified, subcomponents and functions already known, and knowledge sources already identified. For other classes of design this need not be the case. Consequently, class 3 design is an excellent place to start in an attempt to fully understand the complete spectrum of design activity.

Note that we are not merely interested in producing an expert system that produces a trace which is the same as or similar to a designer's, nor are we solely interested in arriving at the same design -- although both are amongst our goals. We are concerned with producing an expert system that embodies a theory of class 3 design and demonstrates the theory's viability.

Imprecision of the Classification: The classification that we have described is a useful way to get a bearing on the complexity of the design task, but it is *not* meant to be formal or rigorous. Neither is the term *routine design*. The approach described in this book is intended to provide a starting point for capturing some of the central phenomena in routine design, but it is not intended to be a complete account of routine design.

References

R. Balzer, "Transformation Implementation: an example," *IEEE Transactions Software Engineering*, SE-7, pp. 3-14, 1981.

D. Barstow, "A perspective on automatic programming," *AI Magazine*, 5,5, 1984.

F. Frayman and S. Mittal, "Cossack: A constraint-based expert system for configuration tasks," II Int. Conf. on Appl. of AI to Engg., Boston, MA, 1987.

P. Freeman and A. Newell, "A model for functional reasoning in design," in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pp. 621-640, 1971.

P. E. Friedland, "Knowledge-based experiment design in molecular genetics," Rep. No. 79-771, Computer Science Dept., Stanford University, 1979. (Doctoral dissertation.)

Adele E. Howe, Paul R. Cohen, John R. Dixon and Melvin K. Simmons, "Dominic: A domain-independent program for mechanical engineering design," *Artificial Intelligence in Engineering*, Vol. 1, No. 1, pp. 23-28, 1986.

Johnson, W. L. and Soloway, E., "Automatic Bug Detection," *Byte Magazine*, April, 1985.

J. Josephson, B. Chandrasekaran, J. Smith and M. Tanner, "A mechanism for forming composite explanatory hypotheses," *IEEE Transactions on Systems, Man and Cybernetics*, Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning, May/June, pps. 445-454, 1987.

Van E. Kelly, "The CRITTER system-automating critiquing of digital circuit designs," *Proceedings of the 21st Design Automation Conference*, IEEE, June, 1984.

A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, 8,1, 1977.

S. Marcus, J. McDermott and T. Wang, "Knowledge acquisition for constructive systems," *IJCAI-85*, pp. 637-639.

D. V. McDermott, "Circuit design as problem solving," in *AI and Pattern Recognition in CAD*, North-Holland, (Ed.), J-C. Latombe, pp. 227-245, 1978.

J. McDermott, "R1: a rule-based configurer of computer systems," in *Artificial Intelligence*, Vol. 19, No. 1, September, pp. 39-88, 1982.

G. A. Miller, E. Galanter and K. H. Pribram, *Plans and the structure of behavior*, New York: Holt, 1960.

T.M. Mitchell, L.I. Steinberg, and J. S. Shulman, "A knowledge-based approach to design," Technical report, LCSR-TR-65, Department of Computer Science, Rutgers University, January, 1985.

S. Mittal and A. Araya, "A knowledge-based framework for design," *Proceedings of AAAI-86*.

S. Mittal, C. Dym and M. Morjaria, "Pride: an expert system for the design of paper handling systems," *IEEE Computer*, 19,7, July, pp. 102-114, 1986.

A. Newell, "Reasoning, problem solving and decision processes: The problem space as a fundamental category," in R. Nickerson, ed., *Attention & Performance VIII*, Erlbaum, Hillsdale, NJ, 1980.

Charles Rich, H. Shrobe and R. C. Waters, "Overview of the programmer's apprentice," *IJCAI-1979*.

E. D. Sacerdoti, "A structure for plans and behavior," New York: American Elsevier, 1977.

R. Schank, *Dynamic memory: A theory of learning in computers and people*, Cambridge University Press, 1983.

R. C. Schank and R. P. Abelson, *Scripts, plans, goals, and understanding*, Hillsdale, N.J.:Lawrence Erlbaum, 1977.

Richard M. Stallman, and Gerald J. Sussman, "Forward reasoning and dependency-directed

backtracking in a system for computer-aided circuit analysis," *Artificial Intelligence*, Vol. 9, No. 2, 1977.

G. Sussman, "A computational model of skill acquisition," Ph.D thesis, MIT Math Dept., 1973.

Appendix H

The Generic Task Toolset

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
March 1987

THE GENERIC TASK TOOLSET

T. Bylander, B. Chandrasekaran and J. Josephson
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Appears in *Proc. Second International Conference on Human-Computer Interaction*.
Honolulu, Hawaii, August, 1987.

The Generic Task Toolset*

Tom Bylander, B. Chandrasekaran, and John R. Josephson
Laboratory for Artificial Intelligence Research, Department of Computer and Information
Science, The Ohio State University, Columbus, Ohio, USA

Abstract

Bylander, T., Chandrasekaran, B., and Josephson, J. R., 1987. The generic task toolset.
Proc. Second Int'l Conf. on Human-Computer Interaction.

Expert system methodologies that emphasize uniformity of representation and the separation of knowledge from the inference engine inherently cannot capture distinctions between different types of problem solving. These methodologies provide no role for higher-level control structures in which knowledge and its use are more closely intertwined, and lack sufficient constraints on the amount of resources available for problem solving. The goal of our research is to study generic types of problem solving that are limited in their expressiveness and computational power, but still provide the basis for explicitly encoding expertise and for much of everyday problem solving by people. This paper briefly describes our research on generic tasks and the development of a toolset for building expert systems using the generic tasks that we have found useful for diagnosis and design.

1. Introduction

Our research in knowledge-based reasoning is characterized by an emphasis on understanding the relationship between problem solving and knowledge in tasks that require expertise. We believe that experts take advantage of several kinds of reasoning in their problem solving, rather than relying on a single inference engine or theorem prover. We also hope that the kinds of reasoning we discover also underlie everyday problem solving by humans. An increasing amount of AI research is being devoted to discovering and studying these kinds of reasoning (Chandrasekaran, 1983, Clancey, 1985). If this research is successful, this could allow for the development of system-building tools that can explicitly represent how an expert uses his knowledge to solve problems.

In this paper, after we describe some of our motivations, we outline our generic task approach, the kinds of reasoning we have studied, and the tools that we have developed. Due to space limitations, we will not be able to present extended examples. The interested reader should turn to the literature cited in the references.

*Research supported by Air Force Office of Scientific Research grant 82-0255, National Science Foundation grant MCS-8305032, and Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010. Toolset development is also supported by grants of equipment from Xerox Corporation, IBM, and Texas Instruments.

2. Motivations Underlying Generic Tasks

The first generation of knowledge-based systems, exemplified by MYCIN (Shortliffe, 1976) and R1 (McDermott, 1982), is characterized by its reliance on production rules. In this methodology, knowledge is thought of as a large collection of facts about a domain (the rules), and reasoning proceeds by applying a domain- and problem-independent inference procedure to those facts (the "inference engine"). Production rules were claimed to have a number of desirable properties (Davis and King, 1977) that need to be carefully questioned.

Uniformity. A uniform representation and inference engine has the advantage of simplicity. There is little overhead in understanding how the basic mechanism operates. However, uniformity results in a level of abstraction problem. A uniform representation cannot capture important distinctions between different kinds of problems. A uniform inference engine does not provide different control structures for different kinds of problems. For example, diagnosis and design are different with respect to the representational distinctions and control structures that they generally call for. Diagnosis generally involves matching observations to malfunctions, setting up a differential, and ruling out competing malfunctions, while design involves synthesizing a structural description, determining the parameters of components and subsystems, and ensuring that design choices satisfy specifications.

Modularity. Although each rule of an expert system may represent a primitive fact or action of a domain, the rules are rarely independent from one another. In order to perform complex problem solving, the rules need to interact with each other, which creates a control problem, i.e., ensuring that the rules interact properly. The need for higher-level control structures for groups of rules has led to programming techniques that create these structures implicitly within the rules. For example, R1, which is implemented in the OPS5 production rule language (Forgy, 1981), performs a sequence of "design subtasks," each of which is implemented as a set of production rules. The programming technique was to encode a condition in each rule that tests the "current context" (the subtask to be performed) and to have rules that switched from one context to another. Thus rules in R1 are not independent actions, but are carefully crafted to interact appropriately with other rules. MYCIN is another example in which implicit structuring took place. Each disease that is represented by MYCIN has an associated set of rules that map data to levels of confidence in the disease. Each rule implicitly depends on other rules to take into account other evidence that affects confidence in the disease and to ensure that backward chaining will search the hypothesis space. Further examples of the use of special programming techniques in rule-based systems are discussed in Buchanan and Shortliffe (Buchanan and Shortliffe, 1984).

Modifiability and Consistency. In view of the above problems, it is difficult to see how rules can be considered easy to modify and easy to check for consistency. The implicit structure of the rules needs to be understood before undertaking any modification or consistency checking on the rules. The uniformity increases the difficulty of the problem by making it hard to determine what type of problem solving is being used.

Separation of the Knowledge Base from the Inference Engine. The general assumption of the first generation of knowledge-based systems was that knowledge has an independent exist-

ence apart from its use -- that knowledge is collected for a given domain, deposited in a knowledge base, and then used by a "knowledge-free" inference engine to solve problems. However, in actual knowledge-based systems, the interactions between the rules and the inference engine could not be ignored.

Resources for Problem Solving. Production rule architectures are equivalent to Turing Machines. This means that any problem that can be solved by a computer problem can be also be solved by some production rule program. Although this generality is useful in terms of its flexibility for programming, production rule architectures, per se, do not distinguish tractable or resource-limited problem solving from intractable or even uncomputable problem solving. Additional constraints are needed to explain how problem solving can be efficient.

3. The Generic Task Approach

Our research has been attempting to resolve these problems by identifying *generic types of problem solving*, which we call "generic tasks" (Chandrasekaran, 1986). Each generic task is a basic combination of a knowledge structure and an inference strategy that are suited for solving a limited type of problem. The idea is to model expert reasoning with several problem solvers, where each problem solver performs a generic task, and all the problem solvers cooperate to solve the problems presented to them. Each generic task is characterized by information about the following:

- The type of *problem* it solves (in terms of the type of input and the type of output). What function does the generic task perform?
- The *representation* of knowledge. How should knowledge be organized and structured to accomplish the function of the generic task? In particular, what are the types of *concepts* and the relationships between concepts that are involved in the generic task?
- The *inference strategy*. What inference strategy can applied to the knowledge to accomplish the function of the generic task?

For example, in the generic task of *hierarchical classification*, the problem is to determine what categories or hypotheses apply to the situation given a description of the situation. The representation is to organize the hypotheses as a classification hierarchy. Each node in the hierarchy can be thought of a high-level module for organizing classification knowledge. Consistency checking can be done separately for each node. The hierarchy is easily modified by adding new nodes or changing relationships between the nodes. The inference strategy (establish-refine) is to consider a hypothesis only if its parent has been determined to be relevant. This is a simple, but powerful strategy for considering only those hypotheses that are relevant to a situation. A significant portion of expert systems such as MYCIN and PROSPECTOR (Duda *et al.*, 1980) can be viewed as hierarchical classification. Other generic tasks that we have identified so far are:

- *Object synthesis by plan selection and refinement* (Brown and Chandrasekaran, 1986). Design an object using hierarchical planning. Design plans are associated with subsystems and components of the object. A design plan might compute the parameters of a subsystem, call for the design of any components, check con-

straints, and handle constraint failures. The tasks performed by the expert systems MOLGEN (Friedland, 1979) and RI can be analyzed in this way.

- *Knowledge-directed information passing* (Mittal *et al.*, 1984). Determine the attribute of some datum based on the attributes of conceptually-related data. For example, a measurement might be abstracted to be low, normal, or high, or one datum might be inferred from another, such as inferring exposure to anesthetics from an episode of major surgery. This generic task is often used in support of other generic tasks such as hierarchical classification or routine design.
- *Hypothesis matching* (Chandrasekaran *et al.*, 1982, Bylander and Johnson, 1987). Match a hypothesis to a situation using a hierarchical representation of evidence abstractions. Different groups of data relevant to a hypothesis are individually evaluated, and then these evaluations are combined into a single judgment on the hypothesis. Samuel's checker-playing program (Samuel, 1967) and the AI RHEUM (Lindberg *et al.*, 1980) expert system perform this task.
- *Abductive assembly* (Josephson *et al.*, 1987). Construct composite hypotheses in order to account for some set of data. An "assembler" selects the "best" (e.g., most plausible) hypotheses that account for the most salient data, parsimonizes composite hypotheses, and critiques them. The INTERNIST (Miller *et al.*, 1982) and DENDRAL (Buchanan *et al.*, 1969) systems largely perform this task.
- *State abstraction* (Chandrasekaran, 1983). Characterize the state of a system based on state changes in its subsystems. Very few expert systems currently specialize in this task.

4. Analyzing Problem Solving in Terms of Generic Tasks

To illustrate how the generic tasks can be used to analyze problem solving, we show how a form of abduction can be performed using generic tasks. We are using the term "abduction" to refer to a form of non-deductive inference that follows a pattern something like this (Josephson *et al.*, 1987):

D is a collection of data (facts, observations, givens),
Hypothesis H is likely or plausible under the circumstances,
H accounts for D (would, if true, account for D).
No other hypothesis accounts for D as well as H does.

Therefore, H is the "best" hypothesis.

That is, abductions go from data describing something, to an hypothesis that best accounts for that data. This form of reasoning is a commonly occurring reasoning problem, e.g., diagnosis, therapy, perception, and theory formation. The functional decomposition described here was used by the RED expert system for red-cell antibody identification (Smith *et al.*, 1985).

A large class of abductive problems can be characterized as a search for the composite hypothesis (e.g., set of diseases, set of characteristics) that best explains a set of observations. Four generic tasks, which interact with each other and with a explanatory facility,

can be used to realize an abductive problem solver. They are hierarchical classification, hypothesis matching, knowledge-directed information passing, and abductive assembly. Their roles are:

- **Hierarchical Classification** - The categories in the classification hierarchy are the hypotheses for the abduction problem. This generic task is used to organize the hypothesis space, filter out the less plausible hypotheses, and explore the more plausible hypotheses.
- **Hypothesis matching** - For each hypothesis in the classification hierarchy, this generic task is used to organize evidence combination knowledge, which is used for estimating the plausibility of the hypothesis.
- **Knowledge-directed data passing** - This generic task is for organizing the observations and to remember which observations need to be explained by some hypothesis. For example in medical diagnosis, symptoms such as pain and bleeding need to be accounted for, while the age and the sex of the patient do not. This generic task is used by the hypothesis matchers as well as the following generic task.
- **Abductive assembly** - This generic task is for generating composite hypotheses. It uses an explanatory facility to determine what data a hypothesis or composite hypothesis can account for. Hierarchical classification is used to generate hypotheses for this generic task, and knowledge-directed data passing is used to store the observations to be accounted for.

5. Conceptualization and Design of the Generic Task Toolset

Each of the generic tasks can be used as a programming technique within a more general programming language like LISP, PROLOG, or OPS5. However, this does not prevent an knowledge engineer from going outside the boundaries of a generic task. These bounds are important to specify and enforce because they ensure that the advantages of generic tasks will be maintained. One natural way to do this is to implement a software tool for each generic task to be used in a general programming environment. Such tools also provide an empirical means for testing the clarity of these ideas and the usefulness of the approach in actual systems.

Consequently, for each generic task, we have developed a tool that can encode the problem solving and knowledge that is appropriate for the generic task. Below is a list of the tools that correspond to the generic tasks that we have studied:

- **CSRL** (Conceptual Structures Representation Language) is the tool for hierarchical classification (Bylander and Mittal, 1986).
- **DSPL** (Design Specialists and Plans Language) is the tool for object synthesis using plan selection and refinement (Brown, 1985).
- **IDABLE** (Intelligent DATA Base Language) is the tool for knowledge-directed information passing (Sticklen, 1983).
- **HYPER** (HYPothesis matchER) is the tool for hypothesis matching (Johnson, 1986).

- *PEIRCE* (named after the philosopher C. S. Peirce) is the tool for abductive assembly of hypotheses (Punch *et al.*, 1986).
- *WWHI* (What Will Happen If) is the tool for state abstraction.

At present each tool has been separately implemented with no attempt at keeping their implementations consistent with each other or facilitating integration of different generic task problem solvers. Part of our current research is to develop a generic task toolset, which integrates all the tools in a consistent fashion, which facilitates the construction of knowledge-based systems that use several generic tasks, and which is implemented in Common Lisp.

The tools are intended to ensure the following advantages of the generic tasks:

- *Multiformity.* Each generic task provides a different way to organize and use knowledge. The knowledge engineer can choose which generic task is the best for performing a particular function, or can use different generic tasks for performing the same function. Different problems can use different generic tasks and different combinations of generic tasks.
- *Modularity.* A knowledge-based system can be designed by making a functional decomposition of its intended problem solving into several cooperating generic tasks, as illustrated in Section 4. Each generic task provides a way to decompose a particular function into its conceptual parts, e.g., the categories for hierarchical classification, and allows domain knowledge of other forms to be inserted into a generic task, e.g., evidence combination knowledge in hierarchical classification (Sticklen *et al.*, 1987).
- *Knowledge Acquisition.* Each generic task is associated with its own knowledge acquisition strategy for building an efficient problem solver (Bylander and Chandrasekaran, 1987). For example in hierarchical classification, the knowledge engineer needs to find out what specific categories should be contained in the classification hierarchy and what general categories provide the most leverage for the establish-refine strategy.
- *Explanation.* This approach directly helps in providing explanations of problem solving in expert systems in two important ways: how the data match local goals and how the control strategy operates (Chandrasekaran *et al.*, 1987). Each generic task localizes the knowledge that is used to satisfy local goals. Also, the control strategy of each generic task is specific enough for generating explanations of why the problem solver chose to evaluate or not to evaluate a piece of knowledge.
- *Exploiting Interaction between Knowledge and Inference.* Rather than trying to separate knowledge from its use, each generic task specifically integrates a particular way of representing knowledge with a particular way of using knowledge. This allows the attention of the knowledge engineer to be focused on representing and organizing knowledge for performing problem solving.
- *Tractability.* Under reasonable assumptions, each generic task generally provides tractable problem solving (Allemand *et al.*, 1987, Goel *et al.*, 1987). One major

exception is abductive assembly, which can become intractable if there are sufficient incompatibility relationships or subtractive interactions between hypotheses.*

The main reasons why they are tractable are because a problem can be decomposed into small, efficient units, and knowledge can be organized to take care of combinatorial interactions in advance.

It should be noted that these advantages are attained at the cost of generality. Each generic task is purposely constrained to perform a limited type of problem solving and requires the availability of appropriate domain knowledge. The taxonomy of generic tasks is far from complete, so a general programming facility will still be needed.

There is also the problem of having to cope with a separate tool for each generic task. A major goal of the toolset is to lessen this burden by providing a consistent interface to all the tools. A similar problem is that the knowledge engineer needs to understand when each generic task is appropriate to use. Thus, the knowledge engineer needs to be trained in performing analyses in these terms.

6. References

- Allemand, D., Tanner, M. C., Bylander, T., and Josephson, J. R. (1987). *On the Computational Complexity of Hypothesis Assembly* (Tech. Rep.). Columbus, Ohio: Lab. for AI Research, CIS Dept., The Ohio State Univ.,
- Brown, D. C. (1985). Capturing Mechanical Design Knowledge. *Proc. 1985 ASME International Computer in Engineering Conference*. Boston.
- Brown, D. C., and Chandrasekaran, B. (1986). Knowledge and Control for a Mechanical Design Expert System. *Computer*, 19(7), 92-100.
- Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A. (1969). Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry. In Meltzer, B., and Michie, D. (Eds.), *Machine Intelligence 4*. Edinburgh: Edinburgh University Press.
- Buchanan, B. G., and Shortliffe, E. H. (Eds.). (1984). *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley.
- Bylander, T., and Chandrasekaran, B. (1987). Generic Tasks for Knowledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition. *International Journal of Man-Machine Studies*, , pp. in press.
- Bylander, T., and Johnson, T. R. (1987). *Structured Matching* (Tech. Rep.). Columbus, Ohio: Lab. for AI Research, CIS Dept., The Ohio State Univ.,
- Bylander, T., and Mittal, S. (1986). CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling. *AI Magazine*, 7(2), 66-77.
- Chandrasekaran, B. (1983). Towards a Taxonomy of Problem Solving Types. *AI Magazine*, 4(1), 9-17.
- Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert*, 1(3), 23-30.
- Chandrasekaran, B., Mittal, S. and Smith, J. W. (1982). Reasoning with Uncertain Knowledge: The MDX Approach. *Proc. Congress of American Medical Informatics Association*. San Francisco: AMIA.
- Chandrasekaran, B., Tanner, M. C., and Josephson, J. R. (1987). Explanation: The Role of Control Strategies and Deep Models. In Hendler, J. (Ed.), *Expert Systems: The User Interface*. Norwood, New Jersey: Ablex.

*The fault is not the strategy of abductive assembly, but because certain kinds of abduction problems are inherently intractable for any computational process.

- Clancey, W. J. (1985). Heuristic Classification. *Artificial Intelligence*, 27(3), 289-350.
- Davis, R., and King, J. (1977). An Overview of Production Systems. In Elcock, E. W., and Michie, D. (Eds.), *Machine Intelligence 8*. New York: John Wiley & Sons.
- Duda, R. O., Gaschnig, J. G., and Hart, P. E. (1980). Model Design in the Prospector Consultant System for Mineral Exploration. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press.
- Forgy, C. L. (1981). *OPS5 Users Manual* (Tech. Rep. CMU-CS-81-135). Carnegie-Mellon Univ..
- Friedland, P. (1979). *Knowledge-based Experiment Design in Molecular Genetics*. Doctoral dissertation. Computer Science Department, Stanford Univ..
- Goel, A., Soundararajan, N., and Chandrasekaran, B. (1987). *Complexity in Classificatory Reasoning* (Tech. Rep.). Columbus, Ohio: Lab. for AI Research, CIS Dept., The Ohio State Univ..
- Johnson, T. R., and Josephson, J. R. (1986). *HYPER: The Hypothesis Matcher Tool* (Tech Rep.). Columbus, Ohio: Lab. for AI Research, CIS Dept., The Ohio State Univ..
- Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C. (1987). A Mechanism for Forming Composite Explanatory Hypotheses. *IEEE Trans. on Systems, Man and Cybernetics*, , pp. in press.
- Lindberg, D. A. B., Sharp, G. C., Kingsland, L. C., Weiss, S. M., Hayes, S. P., Ueno, Y. and Hazelwood, S. E. (1980). Computer-Based Rheumatology Consultant. *Proc. Third World Conference on Medical Informatics*. Tokyo.
- McDermott, J. (1982). R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence*, 19(1), 39-88.
- Miller, R. A., Pople, H. E., and Myers, J. D. (1982). INTERNIST-I, An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine. *New England Journal of Medicine*, 307(8), 468-476.
- Mittal, S., Chandrasekaran, B., and Sticklen, J. (1984). Patrec: A Knowledge-Directed Database for a Diagnostic Expert System. *Computer*, 17(9), 51-58.
- Punch, W. F., Tanner, M. C., and Josephson, J. R. (1986). Design Considerations for PEIRCE, a High Level Language for Hypothesis Assembly. *Proc. Expert Systems In Government Symposium*. McLean, Virginia.
- Samuel, A. (1967). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 11(6), 601-617.
- Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. New York: Elsevier.
- Smith, J. W., Svirbely, J. R., Evans, C. A., Strohm, P., Josephson, J. R., and Tanner, M. C. (1985). RED: A Red-Cell Antibody Identification Expert Module. *Journal of Medical Systems*, 9(3), 121-138.
- Sticklen, J. (1983). *Draft Version Manual for IDABLE* (Tech. Rep.). Columbus, Ohio: Lab. for AI Research, CIS Dept., The Ohio State Univ..
- Sticklen, J., Smith, J. W., Chandrasekaran, B., and Josephson, J. R. (1987). Modularity of Domain Knowledge. *Int'l Journal of Expert Systems: Research and Applications*, , pp. in press.

Appendix I

An Integrated Operator Advisor System

Accepted for publication to Nuclear Technology, January 1990

An Integrated Operator Advisor System for Plant Monitoring,
Procedure Management and Diagnosis.

R. Bhatnagar, D.W. Miller, B.K. Hajen, J.E. Scasenko

The Nuclear Engineering Program
Department of Mechanical Engineering
The Ohio State University
Columbus, OH 43210

Table of Contents

- 0.0 Special Symbols
- 1.0 Introduction
 - 1.1 Organization of the paper
- 2.0 Characteristics of Nuclear power Plant Operation
 - 2.1 Event Oriented Procedures
 - 2.2 Symptom Oriented Procedures
 - 2.3 Integrated Procedures
- 3.0 Generic Tasks and Artificial Intelligence Tools
 - 3.1 The Generic Task of Monitoring
 - 3.2 The Generic Task of Data Abstraction
 - 3.3 The Generic Task of Plan Generation, Execution and Modification
 - 3.4 The Generic Task of Diagnosis
- 4.0 The Architecture of the Operator Advisor System
 - 4.1 The Intelligent Database
 - 4.1.1 Organization of the Database
 - 4.1.2 Data Analysis Methods
 - 4.1.3 Data Inferencing Methods
 - 4.1.4 Queries to Database Classes
 - 4.2 The Plant Status Monitoring System
 - 4.2.1 Knowledge Representation of Safety Functions
 - 4.2.2 Knowledge Representation of Abnormal Events
 - 4.2.3 Plant Data Monitoring

4.3 The Dynamic Procedure Management System

4.3.1 Development of Safety Function Hierarchy for a BWR

4.3.2 Basis of Conflict Resolution and Procedure

Modification

4.3.3 Conflict Resolution

4.3.4 Plan Execution and Modification

4.3.5 Knowledge Representation in DPMS

4.4 Diagnostic and Sensor Validation System

4.4.1 Knowledge Representation in DVS

4.4.2 Functioning of DVS

5.0 Validation and Verification

6.0 Summary and Conclusions

6.1 Summary

6.2 Real-Time System Characteristics

6.2 Future Work

LIST OF FIGURES

- Fig. 1. Overall Architecture of the Integrated Operator Advisor.
- Fig. 2. Overall Data Classification (AllData).
- Fig. 3. Classification of the Continuous Data (SensorData).
- Fig. 4. Classification of Systems and Components (ComponentStates).
- Fig. 5. Classification of Bistable Data (Alarms)
- Fig. 6. Representation of the Sensor TotalReactorFWFlow1.
- Fig. 7. Representation of the System LPCISystemA1.
- Fig. 8. Representation of the Alarm RFEPTripAlarm.
- Fig. 9. Safety Function Hierarchy.
- Fig. 10. List of Abnormal Events.
- Fig. 11. Integrated Hierarchy of Safety Functions and Abnormal Events.
- Fig. 12. Knowledge Representation of LossOffFWHtr Specialist in DVS.
- Fig. 13. Hierarchy of Malfunction Hypotheses used in DVS.
- Fig. 14. Confidence Values for Malfunction Hypotheses With Questionable
Sensor Data.
- Fig. 15. Knowledge Group for Feedwater Recirc Line A Failed Open.
- Fig. 16. Confidence Value for Malfunction Hypotheses After Questionable
Sensor Data Has Been Changed by DVS.
- Fig. 17. Knowledge Representation of the SPECIALIST FWPumpTrip in DFMS.
- Fig. 18. Knowledge Representation of the PLAN for FWPumpTrip Abnormal Event
in DFMS.
- Fig. 19. Knowledge Representation of the PROCEDURE ONI-27 in DFMS.
- Fig. 20. Knowledge Representation of the PLAN for InventoryControl Threat
to Safety in DFMS.
- Fig. 21. Knowledge Representation of the PLAN for RFVControl Threat to

Fig. 21. Knowledge Representation of the PLAN for RPVControl Threat to Safety in DFMS.

Fig. 22. Knowledge Representation of PROCEDURE PEI-B13 in DFMS.

Fig. 23. Knowledge Representation of PROCEDURE PEI-B13-3.0.1 in DFMS.

Fig. 24. Diagnostic Hierarchy below FWPumpTrip in DVS.

List of Tables

Table 1. Safety Functions for A BWR.

Special Symbols

[—] The name within the square brackets is a variable. This variable can be the name of a sensor, parameter, system, component, alarm, state of parameter (i.e. high, low, normal, very high, very low), state of parameter trend (i.e. steady, increasing, decreasing, rapidly increasing, rapidly decreasing), state of system (i.e. available, running, scrambled, operational mode etc.), state of sensor (valid, available), state of alarms (ON, OFF), state of switches (ON, OFF).

1.0 Introduction

Monitoring done by the reactor operator in a nuclear power plant, to maintain normal operation without threatening the safety of the plant is a complex knowledge based task. The complexity of from the complexity of the system and the amount of data required to make decisions. Several computer based operator aids [17, 18, 19, 21] have been developed to provide the data in a more interpreted form to the operator. These aids help the operator only at this level. A more useful operator aid can be of the kind that would look at a large amount of data in a short time, do the data interpretation and in case of failure be able to advise the operator on what abnormal state exists and what actions need to be taken. This kind of operator aid requires the capability to make decisions based on the available data and an understanding of the reactor operation. It is in the development of such aids that the evolving technology of artificial intelligence can contribute.

Several fault diagnostic operator aids [22, 23] of the latter kind have also been developed to assist operators in different areas of malfunction detection. Comprehensive plant diagnostic models have been developed to detect global plant malfunctions. The system developed by Combustion Engineering [24] is capable of diagnosing malfunctions at two levels, a functional level and a root cause level. The system developed by Berg et al. [25] is designed to detect faults well before traditional alarm systems are triggered. Another comprehensive system has been done by the group headed by Naito [26]. This system is designed to diagnose any plant failure and offer (by CRT display) a corrective operational guide for operators in a BWR.

There also are several groups working in the area of procedure tracking

during off-normal conditions. At least two systems have been developed to perform Emergency Operating Procedure (EOP) tracking. One of the approaches, used by Petrick et. al [27], involves a "data-driven" system that monitors the plant status and triggers the EOPs for a BWR-6.

The Reactor Emergency Action Level Monitor (REALM) expert system is designed to provide assistance in the identification of an emergency condition and to determine the severity of the accident [28]. Emergencies are grouped into one of four standard classes by use of Emergency Action Levels (EALs). The four classes of emergencies are: Unusual Event, Alert, Site Area Emergency, and General Emergency.

1.1 Current Research at The Ohio State University

The objective of the current research in the Nuclear Engineering Program at The Ohio State University is to develop an Operator Advisor to assist the human operator in effectively maintaining plant safety. The Operator Advisor being developed is an integrated system having both diagnostic and procedure management components. The diagnostic component is designed to detect faults before the traditional alarms are triggered. The procedure management component is designed to maintain plant safety, providing a defense in depth, by initiating the safety maintenance procedure at the anticipation of threats.

The AI based software, for the Operator Advisor currently under development is based on the framework called Generic Tasks [10,14] proposed by B. Chandrasekaran of The Ohio State University Laboratory for Artificial Intelligence Research (LAIR). The approach proposes that knowledge based systems should be built out of building blocks, each of which is appropriate for a basic type of problem solving. Each generic task provides a higher

level environment which permits increased attention to knowledge representation, problem solving structure and strategy, and specific domain characteristics [9,10,14]. This facilitates knowledge acquisition and organization and makes the system more understandable to the domain experts.

The Operator Advisor consisting of four distinct modules, according to tasks done by a plant operator, has been built. The system consists of (1) an intelligent database to do the task of data interpretation and analysis, (2) a monitoring system to monitor the plant data to detect threats to safety and abnormal functioning, (3) a procedure management system to provide actions to be taken to maintain safety and control the abnormal functioning, and (4) a diagnostic and data validation system to provide further diagnosis to discover the causes of the abnormal plant operations and threats to plant safety.

The knowledge used in the system is based on a Boiling Water Reactor Power Plant. Specifically, the knowledge for data interpretation, monitoring, diagnosis and procedure management is taken from the Operating Manuals of the Perry Nuclear Power Plant.

The Operator Advisor is designed to operate as a real time system and has the following characteristics expected of a real time system [15] :

- . Nonmonotonicity: Incoming sensor data, as well as facts that are deduced do not remain static during the entire run of the program.
- . Asynchronous Events: A real-time system must be capable of being interrupted to accept data from unscheduled or asynchronous events. The events can vary in importance and the system must be capable of processing input according to the importance, even if processing of less important input must be interrupted or rescheduled. The system

should be able to change its focus of attention, according to the importance of the event.

Interface to External Data: System must be capable of gathering data from a set of sensors. The interface also should be capable of extracting features from the data and doing data abstraction.

Temporal Reasoning: System needs to reason about past, present and future events. The system should use time as a resource to do historical analysis of data.

The Operator Advisor system that is being developed has facilities to deal with all these characteristics.

1.2 Organization of the Paper

The next section describes the characteristics of nuclear power plant operation and the existing approaches to safety maintenance these provide the basis for development of the Operator Advisor. This is followed by a description of the Generic Task Methodology for developing expert system tools and expert systems. After this the architecture and the four modules of the integrated system are described. This is followed by a section giving an outline of how the system should be verified and validated. Presented in the conclusion are: the advantages of such a system, how this system deals with the characteristics of a real time system, and the future work required.

2.0 Characteristics of Nuclear Power Plant Operation

In any complex system restoration of normal operation and maintenance of safety are the two required goals when system operation becomes abnormal. However, in general the actions required to achieve these two goals need not coincide. Time spent to analyze and diagnose a malfunction state to restore

operation may degrade the plant conditions to such an extent that safety is compromised. The balance between these two goals may differ from system to system, depending on the understanding of and the knowledge about the functioning of the system and the consequences of threats to safety from the abnormal functioning of the system. In a nuclear power plant the consequences of the threats to safety are far more important than restoring operation in a short time.

The following characteristics of nuclear plant operation provide the basis for the Operator Advisor:

1. Plant operation implies a threat to safety of the environment.
2. Safety cannot be compromised.
3. Safe operation of a nuclear power plant means maintaining the integrity of the core and containment building to prevent release of radiation.
4. The plant can be said to be in a safe state if certain conditions are true, both in the situation of full capacity (or normal operation) or in a situation of abnormal operation (in terms of loss of power, coolant, etc.). The conditions necessary to maintain the plant in a safe state are referred to as safety functions.
5. Safety function maintenance in nuclear plant operation is based on the idea of barrier protection, starting from the inner to the outer barrier i.e. starting from fuel rod to reactor pressure vessel to the containment building.
6. The barrier concept provides a hierarchical relationship between the safety functions, i.e. if a safety function cannot be maintained

because of a procedure not being successful, the safety of the plant can still be maintained by maintaining another safety function, which ensures protection at the next level of the barriers.

7. There can be different kinds of safe states depending on which safety functions are being maintained and which are being threatened. The threat to plant safety varies according to the safety functions being threatened.
8. Normal functioning of the plant implies safe operation
9. The plant can be maintained in a safe state independent of normal operation, because of the existence of independent safety systems.
10. The complete safety of the plant can be maintained by maintaining all the safety functions.
11. Threats to safety maintenance can be identified easily by either a set of parameters that identify the status of safety functions being out of range or by certain alarm conditions. This identification does not require root cause diagnosis and can be done in a short time.
12. The diagnosis of the cause of an abnormal state can be complex and time consuming. The time taken for diagnosis can result in a threat to the safety of the plant.
13. Once threats to safety functions are identified, the safety systems designed to maintain the threatened safety functions are initiated automatically or manually.
14. Two kinds of abnormal states can be defined in the operation of a nuclear power plant: (1) states directly identifiable as threats to safety in terms of entry conditions to any safety function

procedure being true, and (2) states already identified as anticipated incidents (such as abnormal events or causes of alarms) with known entry conditions. The latter kind of abnormal states may not be threatening the safety directly.

15. Both abnormal state categories have procedures defined to control the state. Control in the case of threats to safety means returning the plant to a state, in which the threat is no longer true. Control in the case of abnormal events means mitigating the consequences of the abnormal state to restore operation or to avoid the progress of the consequences that can lead to be a threat to one or more safety functions.
16. Even if the procedure to mitigate the consequences of the abnormal event is not successful, the threat to the safety of the plant can still be avoided by recognizing which safety function will be threatened by the progress of the consequences of the event and then initiating the procedure to control the threat to that safety function or safety functions.

2.1 Event Oriented Procedures

An event oriented approach was used prior to the TMI accident to respond to abnormal occurrences. In this approach, a list of anticipated abnormal events is maintained and procedures to mitigate the consequences of these events are defined. The anticipated events are characterized by a set of specific entry conditions. The procedures for these events also are specific, in the sense that they are related to the specific entry conditions. The procedures are designed to halt the progress of the consequences of the fault. This does not necessarily mean that successful

execution of the procedure brings the plant to a state of normal operation but it does bring the plant to a safe state, from which the plant can eventually be brought to normal operation.

The following weaknesses of this approach were identified by the task force investigating the Three Mile Island accident [30]:

- 1) Too much responsibility was placed on the operator's ability to quickly and properly diagnose the abnormality, and then to respond using the appropriate event oriented procedure,
- 2) The event oriented procedures may lead to incorrect event diagnosis based on predefined event entry conditions,
- 3) Emphasis was placed on maintaining the plant availability while critical safety functions were often ignored,
- 4) Response procedures may lead to a conflict in objectives if multiple events are diagnosed, and
- 5) It is not possible to a priori define all abnormal events or combinations of events.

2.2 Symptom Oriented or Safety Functions Oriented Procedures

This approach evolved to overcome the weaknesses of the event oriented approach. The symptom oriented procedures, or Emergency Procedure Guidelines (EPGs), demand action based on symptoms rather than events. The symptoms associated with these procedures are simple indications in the plant, such as RPV water level low, Drywell pressure high, or reactor pressure high. Even if multiple events occur, following the actions defined for each symptom will assure an operating path that best protects the plant. The idea of having a small group indications as the entry conditions for these procedures eliminated the responsibility of the operator for reaching a

correct fault diagnosis.

This approach is very effective as far as safety of the plant is concerned, but may have an adverse effect on the operation as a heavy cost may be involved in returning the plant to normal operation from a degraded level caused by following the EPG.

A result of the TMI accident was a comprehensive restructuring of the methods for responding to abnormal occurrences [30]. Today, nuclear power plant staffs use a combination of symptom oriented procedures and event oriented procedures, referred to as integrated procedures.

2.3 The Integrated Procedures

This approach uses the event oriented procedures to respond to abnormal anticipated transients. To respond to unanticipated transients, vendor (Westinghouse, General Electric, Combustion Engineering and Babcock & Wilcox) specific symptom oriented procedures have been developed. Some details of the General Electric Emergency Operating Procedures [31] (EOPs) and their integrated use with the Off Normal instructions (ONIs) and Alarm Response Instruction (ARIs) are given in this section.

The EOPs used in GE BWRs, consist of four basic guidelines and six contingency procedures to assure plant safety [29]. These have been customized by each plant to meet their specific configurations. The four basic guidelines for the Perry Nuclear Power Plant (PNPP), designated Plant Emergency Instructions (PEIs), are:

- 1) Reactor Pressure Vessel Control,
- 2) Primary Containment Control,
- 3) Secondary Containment Control, and
- 4) Radioactivity Release Control.

The Reactor Pressure Vessel (RPV) Control Guideline is designed to maintain adequate core cooling and to cool the reactor to a cold shutdown condition. The Primary Containment Control Guideline is used to maintain containment integrity and to protect equipment in the primary containment. Similarly, the Secondary Containment Control Guideline is used to maintain the integrity of the secondary containment and to prevent the release of radioactivity to the secondary containment. The Radioactivity Release Control Guideline is used to limit radioactivity release to the environment. Finally, the contingency procedures provide more detailed instructions for conditions that are more degraded than those providing entry into the basic guidelines.

The above guidelines are further subdivided into the following procedures that control specific parameters of the plant :

RPV Control Guideline

- 1) Reactor Level Control
- 2) Reactor Power Control
- 3) Reactor Pressure Control

Primary Containment Control Guideline

- 1) Suppression Pool Temperature Control
- 2) Suppression Pool Level Control
- 3) Drywell Temperature Control
- 4) Containment Temperature Control
- 5) Drywell Pressure Control

Secondary Containment Control Guideline

- 1) Secondary Containment Temperature Control
- 2) Secondary Containment Pressure Control
- 3) Secondary Containment Level Control

Radioactivity Release Control Guideline

- 1) Radiation Release Control

Each procedure is entered at the beginning whenever any entry condition occurs, irrespective of whether that procedure has been entered before or is currently being executed. The procedure is exited only when an exit

condition is satisfied, or when it is determined that an emergency no longer exists.

The integrated approach is considered to have the advantage of enabling the operator to respond to every plant transient. The event oriented procedures, the ARIs and ONIs, allow rapid recovery of plant operations from specific anticipated plant transients that don't necessarily result in plant shutdown or threats to safety. The EOPs enable the operator to control the effects of any unanticipated plant transient or multiple malfunctions by monitoring simple parameters of the plant. In this approach plant safety is maintained by the EOPs, and thus they receive top priority during execution conflict. That is, regardless of the event, once an EOP entry condition occurs, that procedure is followed.

Corcoran et al. [12] and Meijer [13] used this approach to develop a critical functions maintenance expert system. Neuschaefer et al. [24] have combined this approach with a model based diagnosis system to develop a Generic Diagnostic System.

In our work, the integrated approach for procedure management is further integrated with a diagnostic system based on hierarchical classification (DVS) and a Plant Status Monitoring System (PSMS). This is a further enhancement to the existing integrated approach as a diagnostic component is allowed to determine the cause of the abnormal event or why the threat to the safety function has occurred. The conclusion of the diagnosis is used to determine what actions are needed to correct the cause. These actions are taken without disturbing the procedure being executed to maintain a safety function. This allows the EOPs to dominate the actions taken by the operator. The effects of the actions taken, as a result of the

diagnosis may at a later time lead to exit conditions of the executing EOP. Another enhancement that the diagnostic component provides is the detection of malfunction states before the traditional alarms.

3.0 Generic Tasks and Artificial Intelligence Tools

The concept of generic tasks is a generalization of the idea that lead to writing codes in high level languages, rather than in the assembly language [9, 10, 14]. Engineering design and analysis codes in use for a long time also follow this idea. A special purpose code like CSMP (Continuous System Modelling Program), where the task is the solution of differential equations, is generic in the same way as an AI tool for diagnosis, such as CSRL (Conceptual Structures Representation Language).

CSMP can be used for solving differential equations in any domain such as controls, heat transfer, electrical systems, etc. The user of CSMP has only to provide input in terms of the equations he wants to solve, and is not concerned about how exactly the equations are solved in terms of the lower level language. CSMP, however, can show the details of the solution process in terms of the mathematics of the equation, i.e., a linear equation is being solved with substitution of parameter. This is the level of abstraction at which the engineer operates. He is not concerned with the FORTRAN code written to implement the solution process. If the engineer has to write the code in FORTRAN, then he may lose track of the solution process and become involved with the programming issues. The solution algorithm would appear as FORTRAN code rather than understandable steps used for solution of linear differential equations.

Other generic task codes existing in engineering are SPEAKEZ (a language that allows very high level input for mathematical operations such

as solutions of polynomials), and ACSL (Advanced Continuous Simulation Language, purpose of this is similar to that of CSMP), etc.

The generic tasks required for an operator advisory system can be identified by considering an operator's activities in the plant. The most common activity of the operator is to monitor plant data to determine the state of the plant and to detect any abnormal occurrence. If the state is abnormal and can be indexed to a defined abnormal event, then the operator must initiate the procedure for that event. If the state cannot be mapped to an abnormal event, then the operator must identify whether plant safety is threatened. The safety threat is then monitored while further diagnosis is performed.

The threat to plant safety is identified by threats to safety functions. This is the task of plant status monitoring. This task identifies threats to safety, the predefined abnormal states, and abnormal functioning in general, i.e., any plant data being out of specified ranges.

After the event or the threat to safety has been identified the next task is to execute and monitor the success of the required procedure. During this process, it may be discovered that the procedure is not achieving the desired effects. In this situation, an alternative step or method must be determined to assure that safety barriers are being maintained. This is the task of plan execution.

Another activity that goes on in plant operation is that of diagnosing the lower level fault for the identified malfunction state. This is the task of diagnosis.

These tasks have been implemented in terms of available high level tools. The task of plant status monitoring to identify malfunction states

(threats to safety functions and abnormal events and a general abnormal state) is implemented in the language LOOPS. The task of diagnosis to identify low level faults is implemented as a diagnostic expert system in CSRL [2]. The task of creating a plan to restore operation and safety is done using the tool called DFMS (Dynamic Procedure Management System), developed specifically for this task [4].

A common activity required in all the tasks described is that of data interpretation and analysis. This is implemented through an intelligent database.

3.1 The Generic Task of Data Interpretation and Analysis

This a general task required in all other tasks. The data used for detecting abnormal states, for performing diagnosis, and for evaluating the success of the plans, are very different from the raw data available from process sensors. The task of converting the data to the higher level of abstraction required in the other tasks is done through an intelligent database developed in LOOPS.

3.2 The Generic Task of Monitoring

This is a task distinct from the task of diagnosis done for identification of causes of abnormal states. Monitoring of plant data goes on even when the plant is operating normally. The objective of monitoring is the detection of abnormal data that can lead to identification of any of the two kinds (threats to safety and abnormal events) of malfunction states. The monitoring task also initiates the procedures defined to control the progress of the malfunction state. The decision to carry out diagnosis to identify the low level cause of the malfunction state also is made in the monitoring task depending on the nature of the malfunction (i.e. is there

enough time to do diagnosis or would diagnosis be helpful in a more efficient restoration of operation). In nuclear power plant operation, safety is the priority and the procedures are also safety related. Hence immediately after the identification of a malfunction the procedure for the identified malfunction has to be initiated. Then depending on the identified state, further diagnosis may or may not be required. This decision will depend on how quickly the identified state will threaten safety.

3.3 The Generic Task of Plan Generation, Execution and Modification

In general more than one event or threats to safety functions can be identified by the monitoring system. The first decision in the task of plan execution is that of selecting the best set of plans to be followed. Following the plan set selection, the next task is to initiate the execution of this set of plans. The former task of plan selection is considered a sub task of plan execution and is referred to as Conflict Resolution. This decision is required whenever a new plan is asked to be executed, to decide which plans to remove from the executing set or whether or not to include the identified plan in the executing set.

Before the task of plan execution, the task of plan generation has to be completed. The plan generation task is done through DPMS [4]. DPMS was developed by using the techniques employed in DSPL (Design Specialist and Planning Language). DSPL [8] was developed for the task of design and planning tasks. DPMS is more specific to plan representation and execution of nuclear power plant operating and safety maintenance procedures.

Design as a task is most easily visualized by considering a team of designers whose work is coordinated by the design supervisor [8]. The design supervisor when given initial design requirements and constraints assigns

specific design subproblems to the engineers in accordance with their individual specialities. Through the design process, the supervisor makes decisions on subproblem assignments, ensures that the design subproblems are carried out in the proper order, and calls for redesign if subproblem solutions are not consistent. The supervisor might generate a plan for achieving the design objective, considering what is available and what are the constraints.

The task of plan generation for any objective including the design objective requires the following types of knowledge:

- 1) Knowledge for decomposing the overall plan problem into more manageable plans,
- 2) Knowledge for ordering the execution of these sub plans in a desired sequence,
- 3) Knowledge for testing the success of each sub plan, and
- 4) Knowledge to invoke backup plans to ensure success of the plan goal, if the original plan has not been successful.

DSPL provides agents to represent all these types of knowledge.

In nuclear power plant operation, the objective of the plans is to restore operation while maintaining safety. But in all situations maintaining plant safety is the priority. The restoration of operation cannot be at the cost of safety. The following characteristics of plans for nuclear power plant operation make them different from plans for completing design tasks :

- 1) Predefined procedures are available to bring the plant from initial (abnormal) states to safe (goal) states.
- 2) The environment on which the plans operate is dynamic and the

effects of the plan cannot always be predicted. The executability of the plans has to be determined in run time, based on the existing and changing environment.

- 3) The plans physically change the environment and it is not always possible to undo the effects of a partially executed plan, as would be possible in a conceptual plan.
- 4) The predefined plans must be modified, to at least maintain the plant safety, in a situation where they are not successful, i.e. if the plan and even its backup actions are not executable or successful.

The first characteristic allows the construction of initial plans from existing procedures, thus making the initial planning task easier. The fourth characteristic provides a uniform failure handling mechanism by maintaining safety, and also helps in making the task easier. The second and third characteristics make the task more complex, as a continual evaluation of the partially executed plans is required. It is because of these characteristics that plan modification depends on how much of the standard plan has been executed and what parts of it have been successful.

DPMS was created using the techniques used in DSPL to meet the more specific requirements in plan representation (in terms of language used in nuclear procedures), execution (the need to evaluate the success of the sub plans at runtime before proceeding further), and failure handling (through safety maintenance). The plan execution requires continuous monitoring for success evaluation, which increases the complexity of the task as the environment on which the plans operate is changed and in this case it is not always possible to undo the effects of the executed plan steps.

3.4 The Generic Task of Diagnosis

The diagnostic task starts only when it is known that an abnormal state exists in terms of observed data and process symptoms, and there is a need to determine the lower level cause.

The diagnostic task in nuclear power plant operation starts with generation of a small number of highly plausible hypotheses which account for the observed data [2]. This task is identified as one in which appropriate process symptoms are used to establish or reject hypotheses about malfunctions. The diagnostic process begins with the consideration of a very general malfunction hypotheses about broad segments of the plant. If the process symptoms suggest that a malfunction hypothesis is valid, then the process continues with more refined hypotheses about more detailed subsystems of that particular plant segment. In general; the task, referred to as CLASSIFICATION, is to match a set of symptoms with a specific malfunction hypothesis in a predetermined hierarchical structure of possible hypotheses.

The overall diagnostic task can be viewed as a hierarchy of individual problem solvers, each of which is assigned the duty of establishing or rejecting the relevancy of a particular malfunction hypothesis. Each problem solver contains only the knowledge necessary for the relevancy of a hypothesis.

The classificatory task can then be performed by a top down strategy in which problem solvers high in the hierarchy establish their more general malfunction hypothesis first. If a malfunction hypothesis is established, then the problem solver passes control on to its sibling problem solvers and the process repeats itself.

The classification paradigm using a hierarchical knowledge structure is provided in a high level language called CSRL (Conceptual Structures Representation Language) developed by The Ohio State University Laboratory for Artificial Intelligence Research (LAIR). The language provides a basic structure for hierarchically organizing classificatory concepts, i.e. malfunction hypotheses for organizing knowledge in the problem solvers.

4.0 The Architecture of the Operator Advisor System

The four generic tasks described above form the four modules of the integrated Operator Advisor system [14]. The four modules are:

- 1) An intelligent database
- 2) A Plant Status Monitoring System (PSMS)
- 3) A Dynamic Procedure Management System (DPMS)
- 4) A Diagnostic and Data Validation System (DVS)

Fig. 1 shows the overall architecture of this system. The first two modules of GEPAC Plus and BUFFER are external to the Operator Advisor. GEPAC Plus represents the plant computer which collects the plant data. The data processed by the plant computer will be put in the BUFFER. The database will receive the plant data from the BUFFER.

The database receives the plant data and contains the inferencing knowledge a human operator would use to answer high level questions about data or states of systems or components. In its purpose the database simulates the data abstraction activity performed by the human operator. Just as the control room provides the environment from which the operator gets the data information, the database provides the environment from which the other expert systems get the data information.

PSMS (Section 4.2) performs the task of plant data monitoring to:

1. Identify threats to any safety function,
2. Identify abnormal events that are identified easily by direct conditions (pre-determined entry condition sets or system alarms),
3. Identify the need for further diagnosis,
4. Direct system control either to the diagnostic or to the procedure management sub-systems, and
5. Make further decisions based on the conclusions of the diagnostic system.

The DFMS (Section 4.3) is initiated by PSMS immediately after one or more abnormal events or safety function threats are identified. This system is intended to guide the operator through the procedure defined to mitigate the effects of an abnormal event or to maintain safe operation when safety functions are threatened. The system continuously monitors the success of the procedure, by looking at the updated database, showing the effects of the executed steps. When the defined procedures are not successful, this system provides alternatives to recover from the failed plans or subplans.

The Diagnostic and Data Validation system (Section 4.4) also is initiated by PSMS. This initiation depends on the type of situation. Two types of situations are considered: 1) PSMS is not able to identify an abnormal event or the threat to safety function, but has detected an abnormal situation. The abnormal situation is detected by looking at the deviations from normality. In this situation DVS is initiated at the top node of general fault. 2) PSMS has identified a specific abnormal event. In this situation DVS can be initiated immediately from that specific event, that may be a lower node in the DVS hierarchy. Once DVS is initiated it attempts to find the root cause of the event and communicates its

conclusions to PSMS for further operator action.

In the following four sections a comprehensive description of each of the four modules of the Operator Advisor is given.

4.1 The Intelligent Database

An intelligent database [3] to store plant data and the knowledge to interpret and analyze the data has been implemented. The data and the knowledge available in the database are used to answer high level queries that the operator encounters during diagnosis and procedure monitoring. The database in its content and use is equivalent to the sensors that the operator uses in the plant and the knowledge the operator has gained through experience or from training. An operator's knowledge can be of the following kinds:

1. The criteria to decide what value to use when more than one sensor is measuring the value,
2. The meaning of certain system and component states in terms of plant conditions, for example, what conditions show that the SRV is open besides the direct indication of SRV being open,
3. The data ranges which define the value being normal, high, low, etc, and
4. Temporal information that is available to decide how long some data have been within a certain range.

A facility to update the database as the data changes will be provided to ensure that the Operator Advisor is operating on the current data.

4.1.1 Organization of the Database

The basis of the database organization is centered around objects which represent data types that share the same analysis and inferencing techniques

for producing intermediate data abstractions. This organization provides a means of organizing the techniques in an efficient way that enables similar data types to use a set of relevant techniques.

The database organization starts with a topmost class (AllData) consisting of all data used in the other expert systems. The techniques useful at this level of data are very general and are independent of the nature of the data. Some of the general queries relevant for all data are :

Are all [type of alarms] [on,off] ?

How many [components] in [state] ?

How many sensors are showing a [normal] value ?

The methods used for answering such queries essentially operate on groups of components, systems or sensors of the same kind.

The general class AllData is further subdivided into three classes (Fig. 2) depending on the specific techniques necessary to process the data belonging to the sub classes. These sub classes are:

1. Continuous or Analog (SensorData),
2. States of Component or Systems (ComponentStates), and
3. Alarms, components and systems having two states (Alarms)

The sub class SensorData is specialized into actual process plant parameters. Figure 3 shows some of plant parameters implemented as subclasses of SensorData. The sub class ComponentStates is further specialized into plant systems and sub systems. Figure 4 show some of the subclasses of ComponentStates. The class alarms is specialized into actual alarms used in the plant (Fig. 5).

Each class contains the information, features, and methods needed in analyzing and making inferences. The methods associated with these classes

can also be classified into the following two generic types :

1. Data Analysis Methods
2. Data Inferencing Methods

These types are generic in the sense that they are independent of the expert system task and depend only on the process plant features like sensors, alarms and the ranges and limits of operation. These methods carry out various arithmetic or qualitative operations to get the answer to the high level query.

4.1.2 Data Analysis Methods

The data analysis methods operate on raw data to produce intermediate abstractions. Some of the data analysis methods are:

- 1) Normality Analysis: This determines that a sensor measurement is high, low, normal, very high, or very low.
- 2) Trend Analysis: These determine that an individual sensor reading is steady, increasing, decreasing, increasing rapidly, or decreasing rapidly.
- 3) Redundancy Checking: The abstraction of a single value from multiple common sensors using either maximum value, minimum value or average value (based on the process requirement or state of the plant) of the sensor values.
- 4) Historical Analysis: This determines the changes in values of sensor values or system states with respect to an external measure, i.e. another system reading or system state.

4.1.3 Data Inferencing Methods

Data based inferencing allows information which is not directly

available, to be generated using other data. This information can be abstractions about the functionality of systems, components or process variables. An inferencing of this type uses either available sensor information, diagnostic information, or both to determine unknown data.

4.1.4 Queries to the Database Classes

Continuous or Analog Data

Some of the high level questions that can be asked about data belonging to this class are of the following kind:

- . Is the [process-variable] [high, low, normal, very high, very low] ?
- . Is the [process-variable trend] [steady, decreasing, increasing, increasing rapidly, decreasing rapidly] ?
- . What is the duration of [process-variable value] being [greater, less] than [process-variable value] ?
- . Is the duration of the [process variable value] [greater, less] than [time limit] ?

Fig. 6 shows an instance of this class representing the sensor TotalReactorFWFlow. The slots contain the information necessary to make the analysis and inferencing. Only the slots PresentValue, PreviousValue and Time will be updated by an external interface. The other slots requiring an update will be updated using the values in these three slots and the relevant methods. For example the method for evaluating the trend will be used for updating the Trend slot. In a system where an external preprocessor can decide whether the value is high, low, normal etc., then that decision will be used directly otherwise it will be decided by the database method for Normality analysis.

Component or System States

The high level questions here are in terms of the possible states in which a system or component can exist. Typical questions might be:

- . Is the [component] in [state] ?
- . Is the [system] in [state] ?

The knowledge to answer such questions may involve further inferencing if the answer is not directly available from a sensor. The component, system and state are the variables in these questions. A Component and System can be any component or system defined in the database. A State can be any specific state defined for the specific component or system. Some of the states of the systems or components can be opened, closed, running, available, scrambled etc. Fig. 7. shows an instance of this class. The slots here contain the possible states of the LPCI system and the information required to do the inferencing and analysis. Methods are defined for evaluating each state.

Alarms or Systems having two states

This class is similar to the previous class but is simpler as the systems here can have only two possible states and the information about the states is available directly, without any further inferencing. Typical questions here can be :

- . Is the [alarm] [off, on] ?
- . Is the [light indicator] [on, off] ?
- . Is the [valve] [open, close]?

An instance of this class is shown in Fig. 8. This represents the alarm RFEP (Reactor Feedwater Booster Pump) Trip Alarm. Here the only possible states are OFF and ON and these are directly available without any

inferencing.

The database access is through a query language very similar to natural language. The queries in the other systems are put in this language. The language has a defined grammar which is used by a parser to interpret the queries.

The database is capable of answering questions that use the sensor values available in the database, and can also put a question to the operator if the database discovers that sufficient information is not available to answer a question.

4.2 The Plant Status Monitoring System

The Plant Status Monitoring System (PSMS) is the controlling system that integrates the DPMS and DVS modules using the intelligent database. The system operates in a mode giving priority to the safety of the plant. This system requires the knowledge to detect threats to safety and the abnormal events.

4.2.1 Knowledge Representation of Safety Functions

The safety functions are organized in a hierarchy (Fig. 9) for efficient identification and to provide a basis for maintaining safety in a situation when there are no defined procedures or the defined procedures are not successful. Each safety function node in the hierarchy contains the entry conditions that indicate the threat to safety function. The following relationship exists between the parent and the child nodes in the safety functions hierarchy :

1. A higher level safety function provides coverage of its children safety functions as far as safety is concerned. This is achieved by representing the procedure knowledge in such a way that the

procedure for a parent safety function includes the procedures for its children safety functions.

2. If a safety function can not be maintained then its parent function has to be maintained to ensure safety.

Further details of the basis of development of this hierarchy are described in section 4.3.1.

4.2.2 Knowledge Representation of the Abnormal Events

The abnormal events here consist of all predefined events, abnormal states of all systems and components required for normal operation, and all the alarms and light indicators available in the plant. All these events are stored in a list. The list of events presently available in the system is shown in Fig. 10. Each event in the list contains the entry conditions that establish the existence of the event. The first event in this list (Fig. 10) is a general fault state. The entry conditions or symptoms of the general fault states are the deviations from normality of the important plant parameters. If these deviations exist then it can be said that the plant operation is not normal. This event is essentially the topmost node of the diagnostic system (DVS, described in section 4.4), that indicates Coolant System Fault.

4.2.3 Plant Data Monitoring

PSMS monitors the plant data continuously according to the following cycle :

- 1) Scan the plant data for the entry conditions to threats to any safety functions.
- 2) Scan the plant data for any deviations from abnormality.
- 3) Scan the plant data for the entry conditions to all the

anticipated abnormal events.

In the first step the system does a top down breadth first scan of the safety function hierarchy. Once the entry conditions of a threat to a safety function are true, that safety function is established. At this point PSMS sends a message to the procedure management system (DFMS) to initiate the relevant procedure. The children of the established safety function are not scanned, as it is more important to immediately remove the threat to the parent safety functions, than the children safety functions, and the initiated procedure, for the parent function, will subsequently be able to maintain the children safety functions. After establishing the safety function the monitoring moves to the sibling node of the established safety function.

The second step of monitoring the data for deviations from normality consists of looking at the first event in the list of abnormal events. The entry conditions here are just the deviations from normality. If there are deviations from normality then this malfunction state (i.e. a malfunction state of unknown abnormal functioning) is established. After this PSMS may initiate diagnosis or proceed to look at the rest of the abnormal events in the list (Fig. 10). This decision is based on how much are the deviations from abnormality. If the deviations are major then an abnormal event will be established and hence the system will be asked to proceed to look at the abnormal events. If only minor deviations are present then the system will be asked to proceed with diagnosis through DVS (Section 4.4).

The third step of establishing the existence of any abnormal events is done by looking at the entry conditions of the abnormal events in the list. The event is established if all its entry conditions are true. Once an event

is established PSMS sends a message to DPMS to initiate the corrective procedure. After triggering the procedure, PSMS may also trigger the established event in DVS to find the root cause of the identified event. This decision is dependent on the severity of the event (in terms of threat to plant safety) and the severity of the events remaining to be checked.

After DPMS begins execution and if during the execution of the procedure DVS diagnoses the cause of the malfunction state, then PSMS, initiates another component of DPMS, that contains the plans for the diagnostic causes. This component also will display the actions to be taken. This activity is done without abandoning the execution of the procedure (primary procedure) which may be trying either to restore plant operation or to maintain plant safety. If the actions taken for the cause diagnosed by DVS, are successful, then their consequences may lead to an earlier completion of the primary procedure.

If during the diagnosis DVS finds that a malfunction state was established because of invalid sensor data, then it communicates this information to PSMS, which then sends a message to DPMS to suspend the procedure for that malfunction state. In such a situation it might also be required to reverse the actions taken, or start another procedure if another malfunction state can be established. The reversing of the procedure can be done by including the reversing actions in the DPMS knowledge base. This, however is not done in the present version. The requirement of starting another procedure also is not possible in the present version of DVS, as DVS can only conclude that the malfunction state established because of invalid sensor data should not have been established. It cannot tell which other malfunction state should have been established.

After each cycle of monitoring for detection of threats to safety, deviations from abnormality and the abnormal events, if the systems determines that plant operation is normal then the next cycle begins. If however, the system determines that there is no threat to safety and no abnormal events are true but still the plant conditions indicate that the plant state is not normal, in that, conditions are only partially identifying any threat or abnormal event, or there are deviations from normality, then PSMS will initiate DVS at the top node of the general fault state.

4.3 The Dynamic Procedure Management System

This system is triggered by PSMS after PSMS has identified an abnormal event or a threat to the maintenance of a safety function. As stated in section 3.3.2 the goal of this system is to help in the execution of the required procedure by displaying the procedure steps and then verifying the success of the actions taken. The two subtasks in this system are Conflict Resolution and Plan Execution and Monitoring. Both these tasks are based on the Safety Function Hierarchy.

4.3.1 Development of a Safety Function Hierarchy for a BWR

The development of a safety function hierarchy for a BWR uses the design philosophy of barrier protection and defense-in-depth. The list of safety functions used in DFMS, for a BWR-6, is shown in Table 1.

There are eleven safety functions identified that insure the safe operation of the plant and preserve the public safety. These eleven functions are divided into three safety levels:

- 1) Prevention of Radioactivity Release
- 2) Event Control and Reactor Safety
- 3) Event Prevention and Maintenance of Operations

The hierarchy of the safety functions shown in Fig. 9 was developed [1,5], for a General Electric BWR-6. The lowest level on the hierarchy refers to the event prevention and assurance of proper plant operations. The middle section corresponds to the event containment, and reactor vessel and core safety level. Finally, the highest level corresponds to the prevention of radiation release to the public.

This hierarchy enables the Operator Advisor to estimate the threat of radiation release to the public based on what safety function is being threatened. The overall goal of the hierarchy is to prevent radiation release to the environment. This is achieved by maintaining the top node of the tree. Under normal operations it is considered that there is no threat of radiation release. Once an event occurs that is threatening a node in the hierarchy, then it can be said there exists a potential threat to the top safety function. The degree in which this primary safety function is threatened increases from right to left in the hierarchy.

The lowest level, or event prevention level, emphasizes avoiding situations which would interrupt the generating capabilities of the power plant and the safety systems of the plant. The maintenance of all components in both the safety systems and power generation systems usually guarantees the capability of normal (100% capacity) operations. Some of the safety systems in the plant include: the High Pressure Core Spray (HFCS), Low Pressure Core Spray (LPCS), Low Pressure Core Injection (LPCI), Standby Liquid Control System (SLCS) and all modes of the Residual Heat Removal (RHR) System. Although these systems are not used during normal operating conditions, it is required by the Technical Specifications (TS) to keep each system available. The prevention level also assures the availability of

systems to shut down the plant, to remove heat from the core and to flood the reactor during accident situations.

The power generation systems of the plant are defined as those systems used under normal plant operations. These systems include: the circulating water system, condenser system, feedwater system, pressure regulating system and reactor recirculating system. The operator restores the normal operations of these system through the execution of the procedural steps associated with each node. The maintenance of normal operations also can be described as event mitigation. At this level, mitigation deals with curtailing the damage of other equipment and/or systems as a result of the initiating failure. Corrective actions are taken to maintain the highest operating capacity of the plant.

The role of the second level (or event containment level) is to contain the effects of events that are threatening the reactor pressure vessel or either of the containment barriers. The two critical functions associated with the second level are RPVControl and ContainmentControl. The ContainmentControl function includes maintaining parameters in both the primary and secondary containment structures.

The third level is of prevention of radiation release. If safety is threatened at this level then all resources are directed towards removing this threat.

4.3.2 The Basis of Conflict Resolution and Procedure Modification

This system is based on the integrated event and symptom oriented approach. The integrated approach however has been enhanced in this system by defining the relationship between the event and safety function procedures. Two kinds of relationships are used to select the best plan

(Conflict Resolution) in situations where several plans are asked to be executed, and to provide procedure modification to maintain safety in situations where the procedure along with its backup actions is not successful. First is the relationship between the safety functions (Fig. 9) and the second is the relationship between the event procedures and the safety function procedures [1,4]. This is defined by indexing the abnormal event to a safety function that would eventually be threatened if the effects of the event are not mitigated properly or the event procedure is not successful. This relationship is shown in the presently available augmented hierarchy (Fig. 11). This hierarchy will be expanded in the future to capture the relationship among all the event and safety function procedures.

4.3.3 Conflict Resolution

This is the first task in the total task of plan execution. In general, PSMS may identify more than one threat and more than one abnormal event. The task here is to select the optimal set of plans to be pursued that will best protect the plant in terms of the time taken to reach a safe state. The knowledge used for this selection is the relationship among the safety function and events procedures (Fig. 11). If in this hierarchy, both the parent node and any of its children node are established by PSMS, then because of the way the plans are written from the perspective of safety, it is sufficient to pursue the plan for the parent node.

Conflict Resolution is the task which interrupts the plan execution and updates the set of plans to be followed. When DPMS receives a message of an established malfunction state from PSMS, the controller in Conflict Resolution first checks if the plan for its parent state is active. If it

is, then the plan for the established malfunction is not included in the set of plans to be followed, otherwise it is included. After this check, the controller checks if the plans for any children states are in the set of optimal plans. If they are, then these are removed from the set and the optimal set is redefined and the controller tells the plan execution task to continue execution with the new set of plans.

The controller also controls the optimal set of procedures to be executed in two other situations. 1) In situations when a message from PSMS is received that a plan was initiated because of invalid sensors the controller removes such plans from the set of optimal plans to be pursued. 2) In situations when a message from PSMS is received indicating the diagnostic cause of a malfunction state, as determined by DVS, the controller gives instructions to another component of DPMS to display the set of actions to be taken. In this case the optimal set of plans to pursued is not changed.

It is the task of Conflict Resolution that allows the system to change its focus of attention to take care of: a higher threat immediately, to remove unnecessary plans, and to display corrective actions for root causes. All these three actions provide more efficient execution by avoiding repetition of steps.

4.3.4 Plan Execution and Modification

The task here is to guide the operator through the steps of the identified plan and to continuously monitor the success of the executed steps. The objective is to recover from the abnormal situation by maintaining plant operation and safety, with the priority being on safety. That is, maintain safety even if normal operation is not possible.

Procedures defined for controlling malfunction states have a primary success path available for each anticipated situation, along with alternative success paths. The primary success path and its alternatives may be either for restoring normal operation, or for mitigating the consequences of the malfunction state, depending on the kind of state it is and its relation to safety. If the procedure fails (the primary success path and the alternatives fail) then eventually some safety function will be threatened. The threatened safety function is indexed to the procedure. In this situation the expert system will resort to maintaining safety by initiating the procedure for the indexed safety function, as the system is designed to be safety function maintenance dominant in all situations. This transition to maintenance of safety functions does anticipatory control by alerting the human operator to begin controlling the threat earlier than if he would wait for the entry conditions of threat to safety functions to occur.

The modification of the unsuccessful procedures for controlling the consequences of events or for removing the threats to safety can take place through the following three means :

1. To increase the efficiency of operation, the steps or a sets of steps of the procedures have also been indexed to safety functions. The basis for this indexing is that a step or a set of steps in a procedure are directed at maintaining safety at some level by restoring some safety function. The safety function in some cases will be lower in the hierarchy than the safety function being restored by the overall procedure. This breakup of the procedure can provide better operation by recovering safety at a lower level, requiring lesser effort to restore normal operation

after the threat to safety is removed.

2. If an event oriented procedure is not successful, including its backup actions, then the procedure for the safety function that will eventually be threatened will be initiated.
3. If a safety function procedure is not successful then the procedure will be modified by executing the procedure for its parent safety function in the hierarchy.

4.3.5 The Knowledge Representation in DFMS

The procedures in the operating manuals are represented through three constructs [4], SPECIALIST, PLAN and PROCEDURE, in the knowledge base of this system.

The SPECIALIST is the agent, which initiates the PLAN and provides the final means of failure recovery if failure cannot be recovered at the PLAN or PROCEDURE level. The SPECIALIST is structured as:

```
(SPECIALIST (NAME          )
            (SUPER-SPECIALIST )
            (SAFETY-GOAL      )
            (SUB-SPECIALIST   )
            (CATEGORY         )
            (PLANS             ))
```

In the NAME slot is given the name of the malfunction state (threat to safety function or abnormal event). The information in the SUPER-SPECIALIST and SUB-SPECIALIST slots gives the relationship of the SPECIALIST to other SPECIALISTS. These two slots contain names of the other SPECIALISTS. The SAFETY-GOAL slot specifies the safety goal that will be threatened if the SPECIALIST is not successful. The CATEGORY slot is used to specify the type

of malfunction i.e. whether a threat to safety or an abnormal event. In the PLANS slot the name of the PLAN, to be used to control the malfunction state given in the NAME slot, is given.

The next construct for expressing the procedure is PLAN. The PLAN is structured as:

```
(PLAN (NAME           )
      (SAFETY-GOAL     )
      (EXECUTION-TYPE  )
      (PREREQUISITE    )
      (CRITERIA        )
      (USED-BY         )
      (BODY            ))
```

The PROCEDURE is just a sub PLAN and hence has a structure similar to the PLAN structure containing the same kind of information. The PROCEDURE is structured as:

```
(PROCEDURE (NAME       )
           (SAFETY-GOAL )
           (EXECUTION-TYPE )
           (ACHIEVE-GOAL  )
           (PREREQUISITE )
           (USED-BY      )
           (BODY         ))
```

The NAME slot contains the name given to the PLAN or PROCEDURE in the manuals. In the SAFETY-GOAL slot is stored the safety function that would eventually be threatened if the PROCEDURE or PLAN is not successful. This information is used to maintain safety, in case the PLAN fails.

The EXECUTION-TYPE gives the relationship among the elements of the BODY. Four types are defined SEQUENTIAL, BACKUP, MANYOF, and MONITOR. The most common type is SEQUENTIAL, that is execute the element of BODY sequentially. The BACKUP type implies that the elements of the BODY are alternatives to each other. Each is intended to achieve the same purpose. MANYOF type implies that as many as possible of the elements of BODY should be executed to achieve the best results. MONITOR type means that failure of any elements requires going back and starting from the first element again. The whole plan is not considered successful unless all elements succeed. A fifth type PARALLEL is also possible. The meaning of this is that the elements of BODY should be executed in parallel. This type is not implemented as parallel processes are required for implementing it.

The PREREQUISITE slot prescribes the conditions that should be true before starting the procedure. A common prerequisite would be the availability of the plant systems or components used in the procedure.

The CRITERIA in PLAN and ACHIEVE-GOAL in the PROCEDURE serves the same purpose. They tell what conditions should be true to say that the PLAN or PROCEDURE has been successful.

USED-BY is used to specify which PLANS are using the PROCEDURE. This information is used to exclude failed or executed procedures.

The elements of the BODY are the names of the sub plans (PROCEDURES) and action steps that make up the whole PLAN or PROCEDURE. The following kinds of actions are defined in the system:

1. Actions to be done on systems or components, or actions to be done using systems or components,
2. Conditional actions on or using systems or components,

3. Actions to verify the effects of actions taken, and
4. Actions to monitor the effects of actions for some time.

The first two types of actions modify the environment after they are executed. The second two types of action are just for evaluating the effects of the actions and do not modify the environment.

Modification takes place when a step, procedure or plan fails, and is accomplished through the safety goals. The modification is first tried within the PLAN or PROCEDURE. I.e. the safety goals of procedures and steps are tried so that the rest of the body can be pursued. If this is not possible then the control goes to the SPECIALIST for the failed goal or event PLAN.

4.4 Diagnostic and Sensor Validation System

When activated by PSMS, DVS will look for malfunctions by matching expectations in a malfunction hierarchy. If a malfunction node establishes, DVS will attempt to refine the diagnosis to find the root cause at the lowest available component level.

If a data point is found to be questionable, then the hierarchy will be run again with a different value for the questionable datapoint, this different value will be decided according to values in an expectation pattern stored in the malfunction hypothesis in which the questionable data was detected. It will iterate in this manner until a conclusion is reached and all data are found to be correct. At this point, it will provide DPMS either with confirmation that a proper procedure is being run, or that the malfunction state was established with invalid sensors, and that the procedure should be abandoned.

This system is implemented using the the Conceptual Structures

Representation Language (CSRL). CSRL is a high level language from the Generic Task Tool Kit [7,9], that invokes an ESTABLISH-REFINE [7] control strategy to search the hierarchy of possible malfunction states.

4.4.1 Knowledge Representation in DVS

The malfunction hypothesis in CSRL are represented using the constructs SPECIALIST, Knowledge Groups (KGs) and messages. The SPECIALIST contains the knowledge to establish the malfunction in terms of KGs. The KGs have the knowledge to establish sub hypotheses of the malfunction hypothesis and assign confidence to the sub hypotheses. The knowledge for assigning the confidence to the malfunction hypothesis and what to do next based on the confidence in malfunction hypothesis and its sub hypotheses is given in the construct messages. Figure 12 shows the Specialist LossOffFWHtr used in DVS.

A partial picture of the malfunction hierarchy [2] used in this system is shown in Fig. 13. In this hierarchy of malfunctions, the malfunctions on the right are more general diagnostic classes, and the nodes on the right correspond to more specific cases (type-subtype arrangement of the malfunctions). The set of tip nodes in this hierarchy can be thought of as the diagnostic conclusions relevant to the system.

The data validation routine relies on the pre-compiled knowledge that identifies a discrepancy in the relationship between parameter readings. For example, if a pump sensor is indicating normal operation, then the discharge pressure and the downstream flow sensor should also indicate that flow is present. A sensor is suspected to be erroneous when at least two (and sometime more) sensors are in conflict with its value. The system identifies the specialist in the hierarchy that has used the erroneous data reading by assigning a "*" in its confidence value. The suspected data values are

replaced by the values that are more consistent with other data readings. The new data values are produced through the use of validation methods. These methods contain redundant knowledge that can be used to obtain the new values. The diagnostic system then re-evaluates the previous diagnostic conclusion using the new data value.

CSRL tests malfunction hypotheses in the malfunction hierarchy by first examining the most general nodes, located at the left of the tree, and then moving through the tree at the next lower level from the bottom up. Therefore, it is important to construct the tree with the nodes you want to have considered first at the bottom at each level.

4.4.2 Functioning of DVS [2]

As an example [2,14] of how this system will operate, consider the case of a decreasing water level in the reactor pressure vessel. The specific malfunction is caused by a feedwater recirculation valve inadvertently opening with a concurrent failure of the feedwater level controller calling for no change in feedwater flow rate. To demonstrate how the sensor validation function operates, we also will assume the flow sensor in the recirculation line fails.

A coolant system fault is detected because of the lowering reactor water level. LOCA does not establish because the required expectations, such as increasing drywell pressure (among others), are not present. However, the condition of Reactor Inventory Change does establish. This can be seen in Fig. 14 where a number 3 indicates high confidence (for established hypothesis), and a -3 indicates low confidence (rejection of hypothesis). Knowing the malfunction, one can surmise how each additional node is either established or rejected.

When the given malfunction establishes, it does so with a 3*. The * indicates a possible sensor malfunction. By examining the Knowledge Group for this malfunction in Fig. 15, it can be seen that either the flow rate sensor or the hot surge tank level sensor can cause the *. Because the table is reviewed from the top down, the value for the flow rate is changed in the database, and the hierarchy is run again.

For this case, the result is the same, but without the questionable sensor indication, as shown by the confidence values for the hypotheses in Fig. 16.

5.0 Verification and Validation

The verification and validation of expert systems is different from that of conventional software [16, 33]. This is because in expert systems the search for a solution is much more unbounded than in a conventional system. The search in an expert system is controlled by the knowledge available in the system and the correct functioning is highly dependent on the knowledge it is using. The first task in verification and validation has to be of verifying that correct and consistent knowledge has been represented in the expert system [16, 33].

The verification of expert systems consists of testing whether the software is working as designed and according to the knowledge represented. This activity requires testing of the knowledge and software. Some estimate of the correctness and consistency of the knowledge can however, be made prior to the test runs.

In the Operator Advisor verification is required for (1) the inference knowledge represented in the database, (2) the entry conditions represented in the safety functions and abnormal events, (3) the procedures represented

completely and correctly according to requirements of the grammar defined for procedure representation and (4) consistent knowledge for establishing malfunction hypotheses. Consistency between parent and child malfunction hypothesis in the malfunction hierarchy is essential. Additionally, in the Operator Advisor consistency among the sub systems also is required.

After the consistency of the knowledge represented in the system is verified then the system has to be tested for correct functioning of the software. This is a function of the specifications used for developing the system.

The Operator Advisor was developed from a general set of specifications which stated the broad tasks an operator performs while monitoring the plant during normal steady state conditions and the tasks the operator performs when responding to the deviations from these conditions. The knowledge base and the database have been filled using expert experience and plant data by taking a plant systems approach.

The second factor of Verification and Validation, which is validation of the expert system consists of investigating whether the system actually helps the operator. It is possible that a system functioning as designed may not help the operator. The detailed testing of of this factor can be done only after the system incorporates a large quantity of knowledge and is installed in a real environment. This will require actual operator interaction, which is yet to take place and which is embedded in future plans. This is a two step process:

- (1) Interfacing the Operator Advisor with the plant specific simulator (to accept data directly and respond in real time), and
- (2) Implementing a user (human operator) interface to provide the

information to the operator.

The Operator Advisor has already been tested with a small set of plans for correct functioning according to the knowledge representation. A typical scenario is now described and how the Operator Advisor responded to it referring to the hardcopy output from the system. The hardcopy of the output is showing only the more relevant responses of the Operator Advisor. The details of monitoring and some of the actions encountered in the procedures are not shown in this output. Figs. 17, 18, 19, 20, 21, 22, and 23 show some of the SPECIALISTS, PLANS and PROCEDURES used in the scenario, described in the following paragraphs.

PSMS starts the monitoring by looking at the entry conditions of the safety functions, and from this it has concluded that no safety function is threatened, as no entry conditions were true. Next it has looked at the abnormal events including the general malfunction state. From this monitoring PSMS again has not found any deviations from normality and has not established any abnormal events, hence in this cycle it has concluded that no diagnosis is required.

The second cycle then begins and, in this cycle the system does not established threat to any safety function. In monitoring the abnormal events however, PSMS has established the abnormal event FeedwaterPumpTrip, as all its entry conditions have been found to be true. PSMS has sent a message to DPMS to initiate the procedure for FeedWaterPumpTrip. This is done by triggering the SPECIALIST FeedWaterPumpTripAE-Planner (Fig. 17). After triggering the SPECIALIST DPMS has sent back the acknowledging message to PSMS. The triggered SPECIALIST then initiates the PLAN for FeedWaterPumpTrip (Fig. 18). This plan has only one procedure ONI-27 (Fig. 19). DPMS then

verifies that the procedure is required as its Achieve Goal is not true. It then confirmed that the procedure can be started as its Prerequisites are available. After this the required actions are displayed and verified for success.

During the execution of the procedure (ONI-27) for FeedwaterPumpTrip (Fig. 19), however, PSMS has initiated the diagnosis for the FeedWaterPumpTrip node in the DVS hierarchy (Fig. 13). The hierarchy below FeedWaterPumpTrip is shown in Fig. 24. DVS has diagnosed the cause of FeedWaterPumpTrip as Low NPSH. This conclusion is communicated to DPMS. After receiving this conclusion DPMS displays the instructions (TAKE CARE OF LOW NPSH) and continues with the procedure for FeedWaterPumpTrip.

During further execution of the procedure for Feed Water Pump Trip, PSMS has established the threat to the higher safety function of InventoryControl. This message is communicated to DPMS. DPMS has acknowledges the message and shifts its operation to maintenance of InventoryControl (Fig. 20). The system has not been able to execute the procedure for InventoryControl successfully as the actions taken have not yielded the required results. DPMS has started failure recovery by shifting the operation to maintenance of the higher level safety function of RPVControl (Fig. 21).

DPMS initiates the PLAN for RPVControl and starts the execution. This plan contains the procedure PEI-B13 (Fig. 22). Fig. 23 shows one of the sub procedures PEI-B13-3.0.1 of PEI-B13 The further output shows the actions and success monitoring of the RPVControl procedure. The procedure is eventually successful and the system has been able to maintain safety at the level of RPV Control.

The scenario described above is typical of tests of the system for specific plans. Before initiating the validation stage a large number of plans must be tested. The system is however, for the scenario described, behaving as expected for the plans used in the scenario.

6.0 Summary and Conclusions

6.1 Summary

The final product of the research described in this paper is a framework for an Integrated Operator Advisor. The framework can be functional only after it incorporates a large amount knowledge. After the required knowledge is incorporated then it is expected that the Operator Advisor will be able to aid the operator in the following:

- (1) Identifying malfunction states,
- (2) Providing the procedure for the malfunction state,
- (3) Monitoring the success of the procedure,
- (4) Providing safety maintenance in situations when the prescribed procedure fails or there is no procedure available,
- (5) Providing the cause of the malfunction state by diagnosis,
- (6) Giving instructions for rectifying the cause,
- (7) Changing the focus of attention if a higher threat occurs while pursuing a procedure,
- (8) Detecting malfunction states through DVS, before the traditional alarms, and
- (9) Anticipating the threat to safety and initiating the required safety maintenance procedure.

6.2 Real-Time System Characteristics

The Operator Advisor has the characteristics [15] summarized in the

introduction of real-time systems and has facilities to handle them efficiently.

- . Nonmonotonicity of data: The change in plant conditions and data is taken care of by continuously monitoring the data and the success of the actions taken.
- . Interface to the External Environment and Data Abstraction: The intelligent database is capable of taking data from a set of sensors and performing feature extraction and data abstraction. These activities are done at runtime whenever required.
- . Asynchronous Event and change in Focus of Attention: The controller in the conflict resolution task is specifically designed to interrupt ongoing tasks. The controller allows change of focus of attention in the following situations:
 - 1) If during the execution of a procedure, a higher threat is recognized by PSMS, then DPMS will accept this message and make a decision based on the threat's relation to the executing procedure. If the threat is higher, DPMS will change its focus of attention and the procedure for the higher threat will be initiated.
 - 2) In the situation when DVS discovers the cause of a malfunction state, then DPMS accepts this message and stops the execution of the active procedure for some time to give instruction to correct the cause. After giving these instructions DPMS continues with the earlier active procedure and checks for its success from the present conditions, which might have changed due to

actions taken for the cause.

- 3) The system is also capable of changing its focus depending on the situation at runtime, e.g. if a plan is failing then the plan for its safety goal will be initiated, by trying to maintain safety.

- . Temporal Reasoning: The database uses time as resource to answer historical questions about data, such as For how long has the Feed water flow been greater than the steam flow ?. The EXECUTION-TYPE monitor also uses time to evaluate the success of the PROCEDURE. The future actions on procedure failure are taken based on the future consequences of uncontrolled event.
- . High performance: The system is complete and robust from the perspective of safety, as it can lead the plant to a safe state in all situations, by maintaining safety at a greater depth.

6.3 Future Work

Several enhancements and improvements of the system are possible that can make it more robust. Following are some of issues for future work :

- (1) Addition of more knowledge in the four modules.
- (2) Replacing the database with a commercial database. The commercial database would provide better update and management facilities.
- (3) Additions in the DVS system to go down to specific sensor validation, presently it does only a query validation.
- (4) Additions in DPMS to have the facility to reverse the actions of the procedure not required. This can be done by providing reversing procedures whenever possible and can be achieved by additional knowledge for certain procedures and may not require modifications or

additions in the software.

- (5) Automate the decision to start diagnosis of a malfunction state or continue with monitoring. This would require some addition to the software but can be done only after the knowledge to make this decision is available.
- (6) Implement the system on a multitasking machine to facilitate several parallel processes. The parallel processes that are required in the system are: (1) background process to ask for database update (2) Plant data monitoring process (3) Procedure monitoring process (4) Diagnosis process (5) Parallel processes to implement the parallel actions requirements in procedure (6) Process to monitor the procedure for rectifying the cause (discovered by DVS) of malfunction states.
- (7) Install the system on a simulator for testing the software and knowledge base [32].
- (8) Add better graphic displays and human interface facilities.

The testing in a full function simulator is a key step in the validation of the Operator Advisor. The planned testing program will involve plant operator evaluation over an extended period of time with a variety of test scenarios. The comparative response of plant operators with and without the Operator Advisor will provide insight into the value of the Operator Advisor to the plant operators.

Prior to installation, the plant operational staff will be encouraged to provide input in design of the Operator Advisor especially those facets effecting human interface. During and after the evaluation by the staff it is expected they will continue to provide valuable input to improving the overall conceptual approach as well as further design enhancements.

Acknowledgments

The authors would like to acknowledge the contributions of B. Chandrasekaran, D. D. Sharma, N. Yamada, W. F. Punch and S. Hashemi.

The research presented in this paper has been partially supported by two grants from the National Science Foundation and a grant from the Department of Energy. Specifically the initial conceptualization and the initial development of the overall diagnosis and data validation sub systems, and the procedure management prototype was supported by NSF grant No. CBT 8400840. The integration of the two prototypes through the intelligent database and the plant status monitoring sub system plus initial verification and validation has been supported by DOE grant No. DE-AC02-86NE37965. Finally, the conceptual aspects of temporal reasoning and real-time systems has been partially supported by a second NSF grant No. ECS 8612254.

The authors would like to express their appreciation to the staff of the Perry Nuclear Power Plant Nuclear Training Department for their extensive assistance in obtaining plant data and for working with us to run plant transients on the Perry Plant simulator. This collaboration has provided a reference plant for the research described in this paper and without it, the progress made would not have been possible.

The collaboration of GE Nuclear Energy, Electronic and Computer Services, also is acknowledged and appreciated. GE's collaboration is making it possible to assure that our expert system will properly interface with the plant computer systems.

Finally, the authors wish to acknowledge the support provided by The Ohio State University. Specifically the Laboratory for Artificial Intelligence Research, the Nuclear Engineering Program, and the Department of Mechanical Engineering.

Table 1 Safety Functions for A BWR

Safety Function	Level	Purpose
Prevent Radiation Release	1	Protect Public From Radiation
Containment Pressure Control Containment Temperature Control	2	Prevent Containment Barrier Damage
Drywell Pressure Control Drywell Temperature Control	2	Prevent Damage to Drywell Barrier and Equipment
Suppression Pool Level Control Suppression Pool Temperature Control	2	Maintain Availability of Emergency Heat Sink
Reactor Inventory Control	2	Maintain Coolant Around Core
Reactor Pressure Control	2	Maintain Reactor Vessel Pressure
Heat Sink Control	2	Assure Proper Heat Removal from Core Coolant
Maintain Vital Systems	3	Maintain Operation of Systems needed to Support Operations

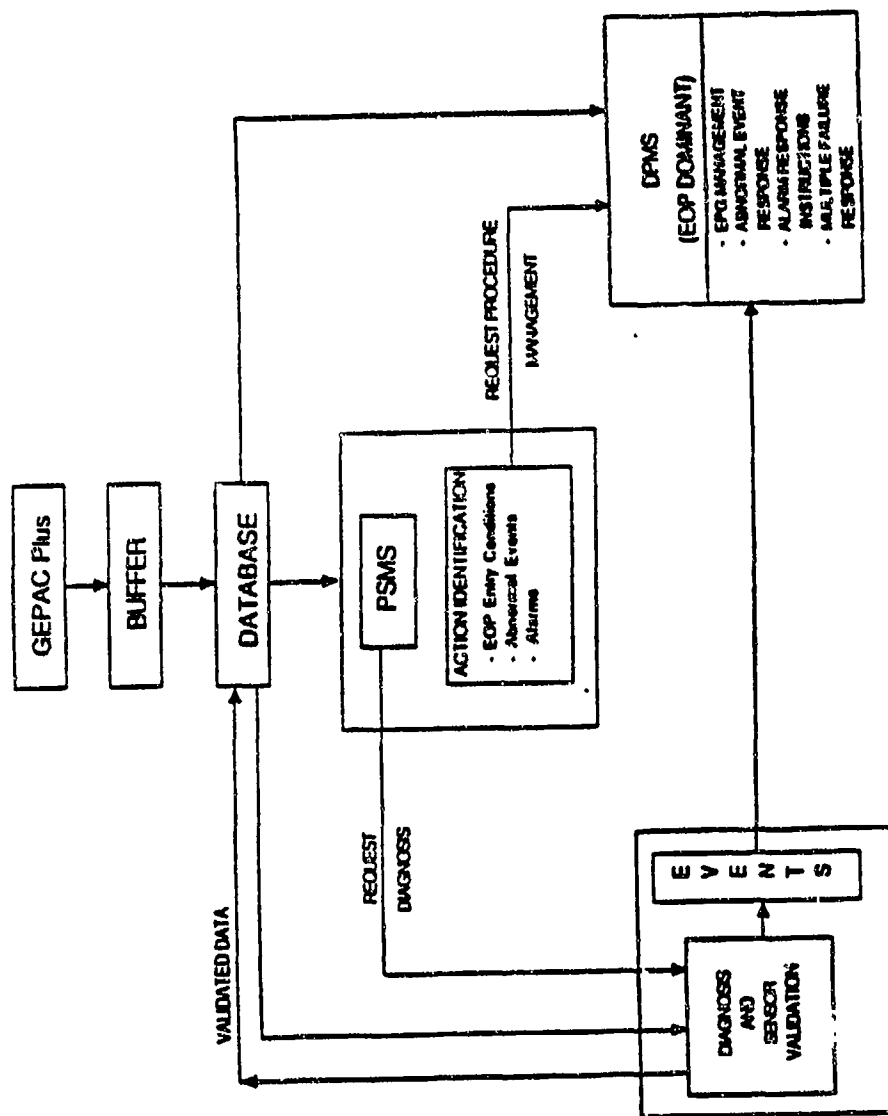


Fig. 1. Overall Architecture of the Integrated Operator Advisor.

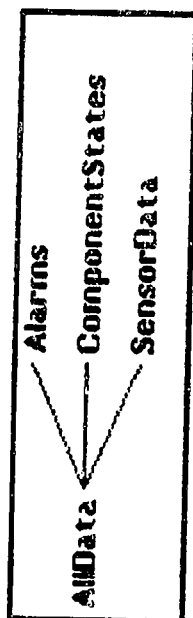


Fig. 2. Overall Data Classification (AllData).

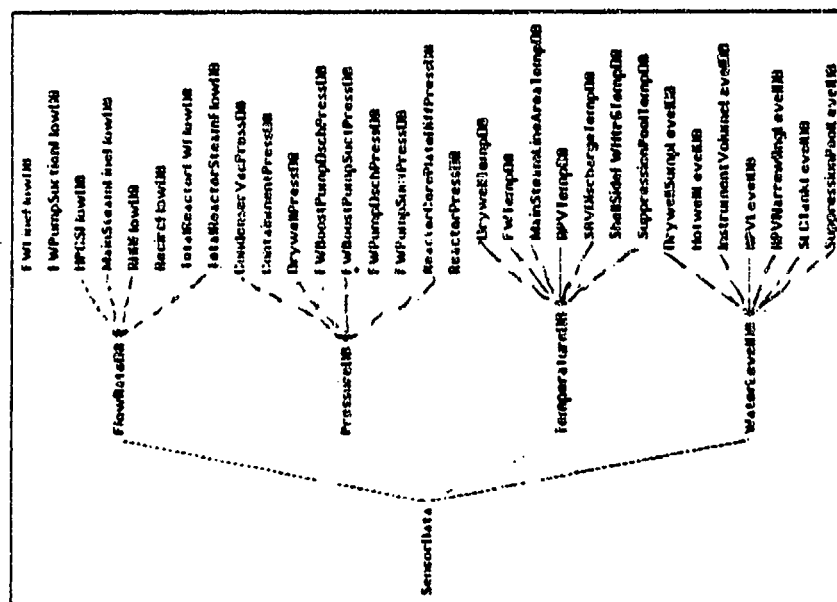


Fig. 3. Classification of the Continuous Data (SensorData).

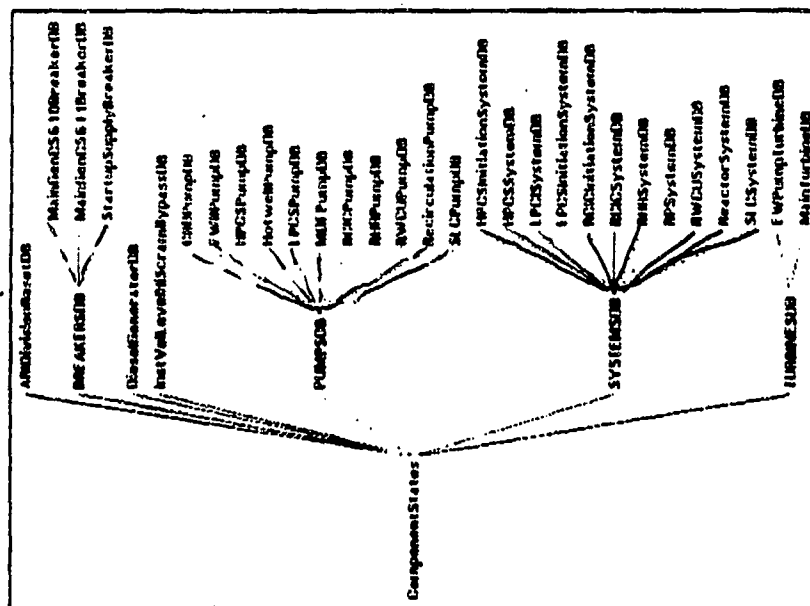


Fig. 4. Classification of Systems and Components (ComponentStates).

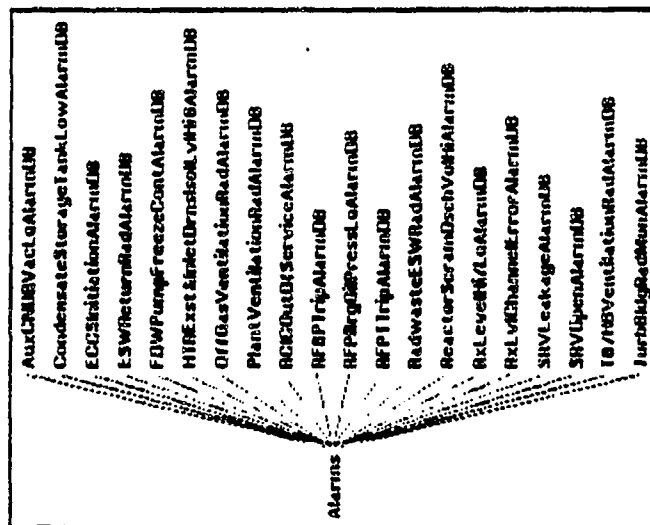


Fig. 5. Classification of Bistable Data (Alarms)

DataName	-Total Reactor FWF low (C34-Huc2A88)	
OperatingCondition	?	
SensorUsedInSpecification	?	
Time	?	
NormalValue	?	
HiLimit	20	
LoLimit	0	
HiActionLimit	?	
LoActionLimit	?	
HiAlarmLimit	?	
LoAlarmLimit	?	
HiNormalRange	?	
LoNormalRange	?	
Trend	0	
PreviousValue	16	
QPreviousValue	L	
PresentValue	16	
QPresentValue	L	
Frequency	?	
Interval	?	
DurationSet	?	
HiDurationSet	?	
LoDurationSet	?	
HiPercentageDuration	?	
LoPercentageDuration	?	
HiLimitInit	?	
LoLimitInit	?	
HiDuration	?	
LoDuration	?	
ComputerID	C34EA019	

Fig. 6. Representation of the Sensor TotalReactorFWFlow1.

OperatingCondition	T	"LPCI SYSTEM LOOP-A"
DataName		
SensorUsedInSpectralist		(Used in DPMS)
Opened	T	
Closed		NIL
Isolated	T	
Tripped		NIL
Shutdown		NIL
Auto	T	
Manual		NIL
Available	T	
Running		NIL
Operating		NIL
ComputerID		"PUT POINT ID"
Initiated		NIL

Fig. 7. Representation of the System LPCISystemA1.

DataName	"RFBP-A trip alarm"
ScanInstance	0
OperatingCondition	NIL
OFF	NIL
ONN	H
SensorUsedInSpecialist	NIL
frequency	NIL
Interval	10
ComputerID	"Computer Point 10"

Fig. 8. Representation of the Alarm RFBPTripAlarm.

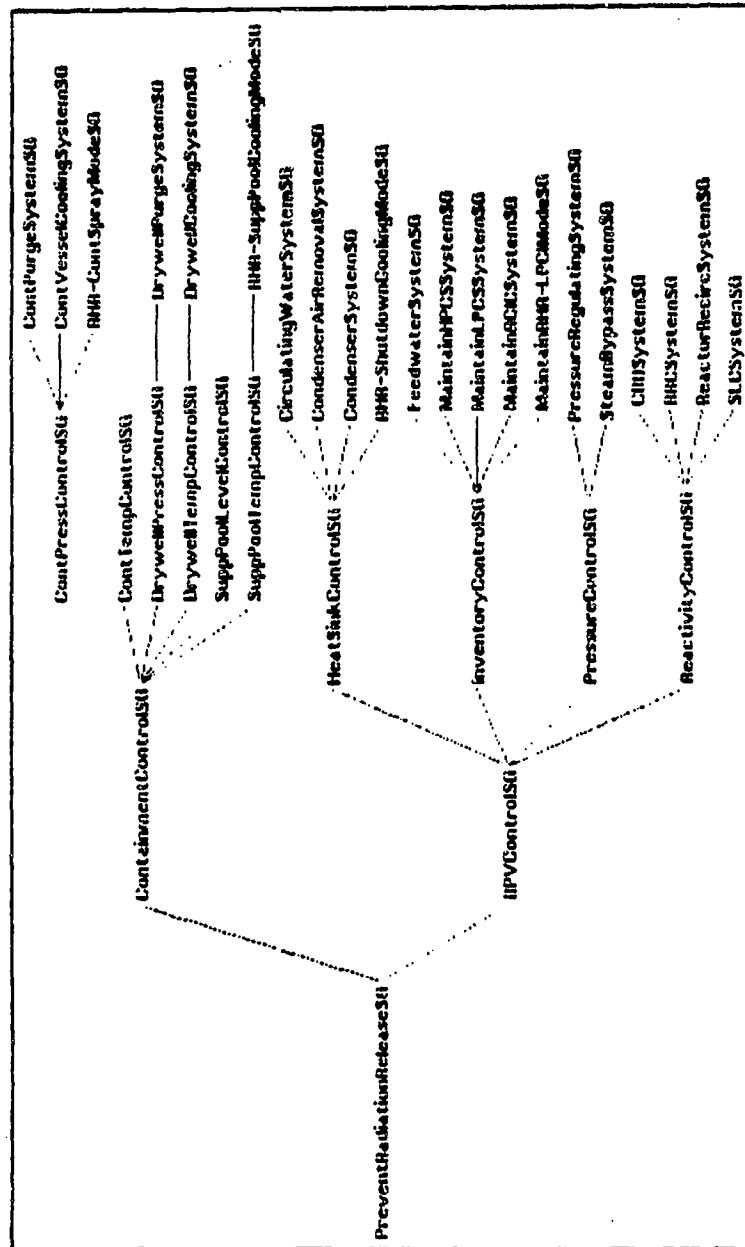


Fig. 9. Safety Function Hierarchy.

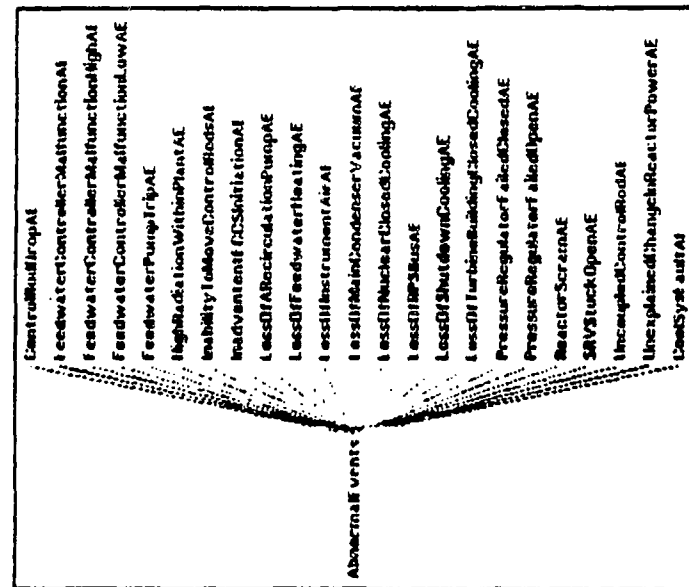


Fig. 10. List of Abnormal Events.

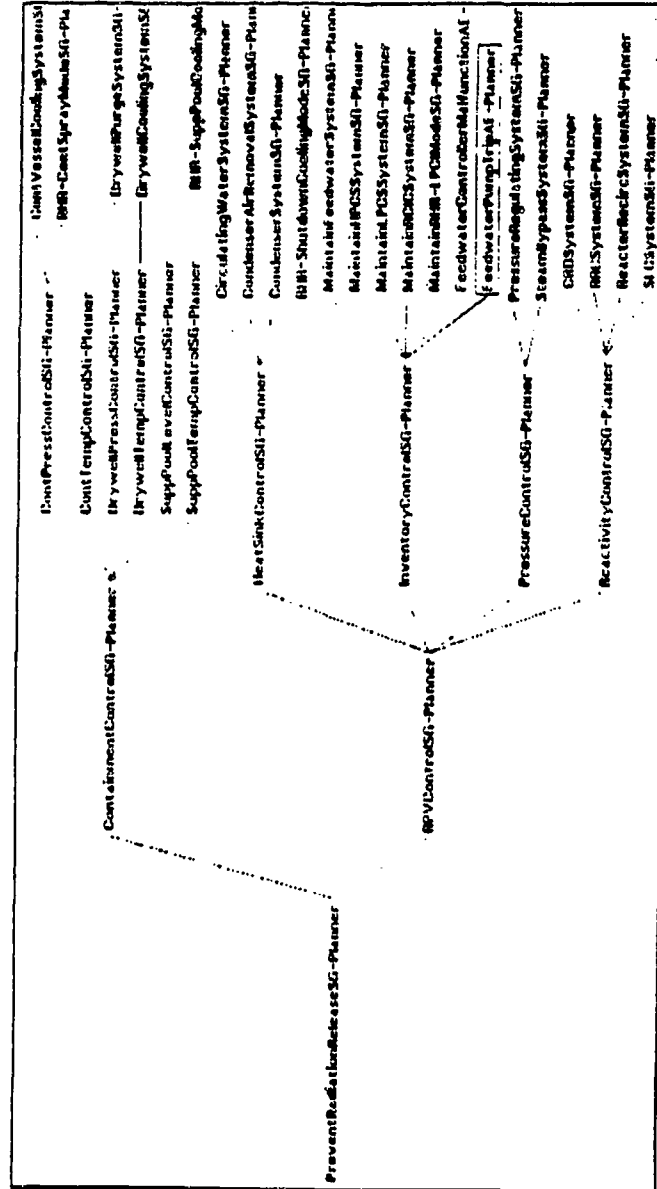


Fig. 11. Integrated Hierarchy of Safety Functions and Abnormal

Events.

```

(Specialist
LossOfFWHr
(declare (LoopSuper NPPSpecialist)
(SuperSpecialist ColdWaterAcc))
(kgs (Water&SteamFlow

Table
(match (AskHLN? "What is the steam flow rate")
(AskHLN? "What is the steam flow trend")
(AskHLN? "What is the feedwater flow rate")
(AskHLN? "What is the feedwater flow trend")
with
(if N S H S
then 3
elseif N ? H ?
then 2
else -3)))

(Reactor

Table
(match (AskHLN? "What is the feedwater temperature")
(AskHLN? "What is the reactor power level trend")
(AskHLN? "What is the reactor water level trend")
(AskHLN? "What is the feedwater temperature trend")
with
(if (OR N L LL)
(OR I II)
(OR S I II)
(OR D DD)
then 3
elseif (OR H HH)
(OR I II)
(OR S I II)
(OR D DD)
then 3
else -3)))

(messages (Establish
(if (GT Water&SteamFlow 1)
then (if (EQ Reactor (QUOTE 3))
then (doList (* ($ currentCase)
StoreSusData SPECIALIST KName
CaseNo)
(SPECIALIST self)
(KName (QUOTE Reactor))
(CaseNo ($ currentCase)))
else (SetConfidence self Reactor))
else (SetConfidence self Reactor))
else (SetConfidence self Water&SteamFlow))))

```

Fig. 12. Knowledge Representation of LossOfFWHr Specialist In

DVS.

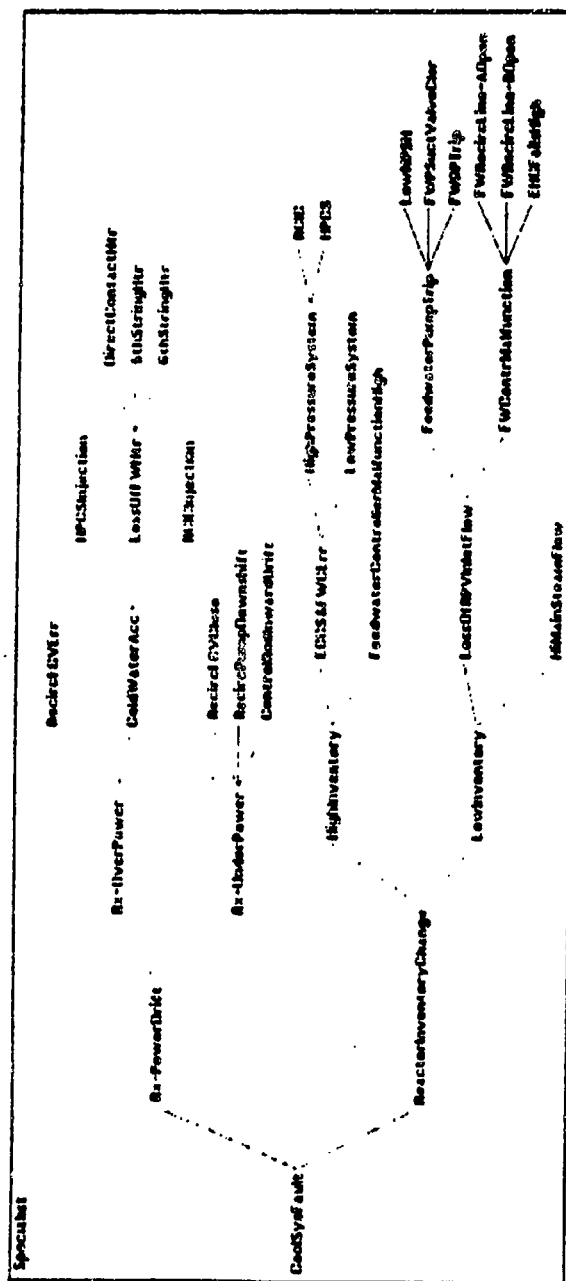


Fig. 13. Hierarchy of Malfunction Hypotheses used in DVS.

FWRecirc-A kg of FWRecircLine-AOpen									
Expressions of table									
1 - (+ (\$ FWPumpSuctionFlowA1)	?	?	?	?	?	?	?	?	value
SensorDataValue	H	H	H	H	H	H	H	H	-3
(QUOTE QPresentValue))									-2
2 - (+ (\$ FWLineFlowA1)	(OK H H HH)	(OR L LL)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	3+
SensorDataValue	(OK H H HH)	(OR L LL)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	3
(QUOTE QPresentValue))	(OK H H HH)	(OR L LL)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	3+
3 - (+ (\$ FWLineFlowA1)	?	?	?	?	?	?	?	?	?
SensorDataValue									
(QUOTE QPresentValue))									
4 - (+ (\$ FWLineFlowA1)	(OK H H HH)	(OR L LL)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	3+
SensorDataValue	(OK H H HH)	(OR L LL)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	3+
(QUOTE QPresentValue))	(OK H H HH)	(OR L LL)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	(OR H H HH)	3+
5 - (+ (\$ HotSurgeTankLevel1)	?	?	?	?	?	?	?	?	?
SensorDataValue									
(QUOTE Trend))									

Fig. 14. Confidence Values for Malfunction Hypotheses With Questionable Sensor Data.

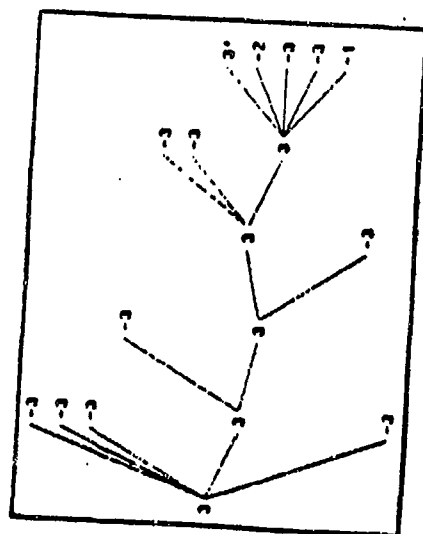


Fig. 15. Knowledge Group for Feedwater Recirc Line A Failed Open.

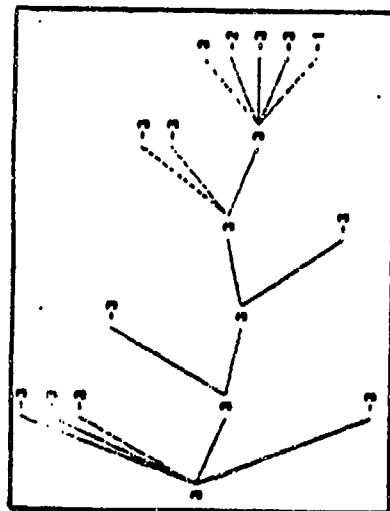


Fig. 16. Confidence Value for Malfunction Hypotheses
After Questionable Sensor Data Has Been
Changed by DWS.

(Specialist (NAME FeedwaterPumpTripAE-Planner)
(SAFETY-GOAL InventoryControlSG-Planner)
(SUPER-SPECIALIST InventoryControlSG-Planner)
(SUB-SPECIALIST NONE)
(CATEGORY NONE)
(PLANS FeedwaterPumpTripAE-Plan))

Fig. 17. Knowledge Representation of the SPECIALIST FW_{PumpTrip} in DEMS.

(PLAN (NAME FeedwaterPumpTripAE-Plan)
 (SAFETY-GOAL InventoryControlSG-Planner)
 (EXECUTION-TYPE Sequential)
 (CRITERIA (OR (AND (IsTheNumericalValueOfRPVLevelDB > 197 inches)
 (IsTheNumericalValueOfRPVLevelDB < 205 inches)
 (AND (IsTheNumericalValueOfReactorPowerDB > = 26 percent))
 (IsTheNumericalValueOfRPVLevelDB > 178 inches)
 (IsTheNumericalValueOfReactorPowerDB < 4 percent))
 (PREREQUISITE NONE)
 (USED-BY FeedwaterPumpTripAE-Planner)
 (BUILT ONI-N27))

Fig. 18. Knowledge Representation of the PLAN for FW Pump Trip

Abnormal Event in DWS.

```

(PROCEDURE (NAME ONI-N27)
  (SAFETY-GOAL InventoryControlSG-Planner)
  (ACHIEVE-GOAL (AND (IsTheDurationOfRPVLevelDB > 197 inches for > 10 mins)
    (IsTheDurationOfRPVLevelDB < 205 inches for > 10 mins)))
  (PREREQUISITE NONE)
  (EXECUTION-TYPE Sequential)
  (USED-BY FeedwaterPumpTripAE-Plan)
  (BODY ONI-N27-3.0 ONI-N27-4.0
    (MONITOR (CanMaintain RPVLevelDB > 197 inches for > 10 mins)
      THEN
        (DISPLAY (The RPV level has been > 197 for > 10 mins)))
    (MONITOR (CanMaintain RPVLevelDB < 205 inches for > 10 mins)
      THEN
        (DISPLAY (The RPV level has been < 205 for > 10 mins)

```

Fig. 19. Knowledge Representation of the PROCEDURE ONI-N27 in

DFMS.

```

(PPLAN (NAME RPVControlSG-Plan)
(SAFETY-GOAL PreventRadiationReleaseSG-Planner)
(EXECUTION-TYPE Sequential)
(CRITERIA (AND (CanMaintain RPVLevelDB > 178 inches for > 10 mins)
                (CanMaintain RPVLevelDB < 215 inches for > 10 mins)
                (IsTheNumericalValueOfDrywellPressDB < 1.68 psig)
                (IsTheNumericalValueOfReactorPressDB < 950 psig)))
(PREQUISITE NONE)
(USED-BY RPVControlSG-Planner)
(BODY PEI-B13))

```

Fig. 20. Knowledge Representation of the PLAN for
InventoryControl Threat to Safety in DEWS.

(PLAN (NAME InventoryControlSG-Plan)
(SAFETY-GOAL RPVControlSG-Planner)
(EXECUTION-TYPE Sequential)
(CRITERIA (IsTheNumericalValueOfRPVLevelDB > 178 inches))
(PREREQUISITE NONE)
(USED-BY InventoryControlSG-Planner)
(BODY InventoryControlProcedure))

Fig. 21. Knowledge Representation of the PLAN for RPVControl

Threat to Safety in DPM.

```

(PROCEDURE (NAME PEI-B13)
  (SAFETY-GOAL RPVControlSG-Planner)
  (ACHIEVE-GOAL (AND (CanMaintain RPVLevelDB > 185 inches for > 10 mins)
    (CanMaintain RPVLevelDB < 215 inches for > 10 mins)
    (IsTheNumericalValueOf ReactorPressDB < 950 psig)
    (IsTheNumericalValueOf ReactorPowerDB < 4 percent)))
  (PREREQUISITE NONE)
  (EXECUTION-TYPE Sequential)
  (USED-BY RPVControlSG-Plan)
  (BODY PEI-B13-3.0.1 ONI-C71
))

```

Fig. 22. Knowledge Representation of PROCEDURE PEI-B13 in DFMS.

```

(PROCEDURE (NAME PEI-B13-3.0.1)
  (SAFETY-GOAL InventoryControlSG-Planner)
  (ACHIEVE-GOAL NONE)
  (** No ACHIEVE-GOAL is specified due to the fact that the conditions are check in
the
  procedure PEI-B13)
  (PREREQUISITE NONE)
  (EXECUTION-TYPE Sequential)
  (USED-BY PEI-B13)
  (BODY (Check if the reactor scram has initiated)
    PEI-B13-3.1.1-MONITORA
    (IF (ASKOPERATOR (AreAll RPSChannelScramStatusDB Scrammed))
      (** These are the conditions that characterize a reactor scram
      (AND (AreAll ControlRodPosDB <= 2)
        (AreAll ScramValveDB Opened)
        (IsTheNumericalValueOf ReactorPowerDB < 4 percent)
        they are check in the method for Reactor Scrammed))
      THEN
        (SET ReactorModeSwitch1 to Shutdown)
      ELSE
        (Arm and Depress RPS Manual Scram CH A-D pushbuttons)
        (SET ReactorModeSwitch1 to Shutdown))
    PEI-B13-3.0.3 PEI-B13-3.1.1-MONITORB (* This is the beginning of the PEI
RPVControl
                                         steps 3.0.1 and 3.0.2 on page 4)))
                                         will be removed from
                                         RPVControl)))

```

Fig. 23. Knowledge Representation of PROCEDURE PEI-B13-3.0.1 in
 DEFS.

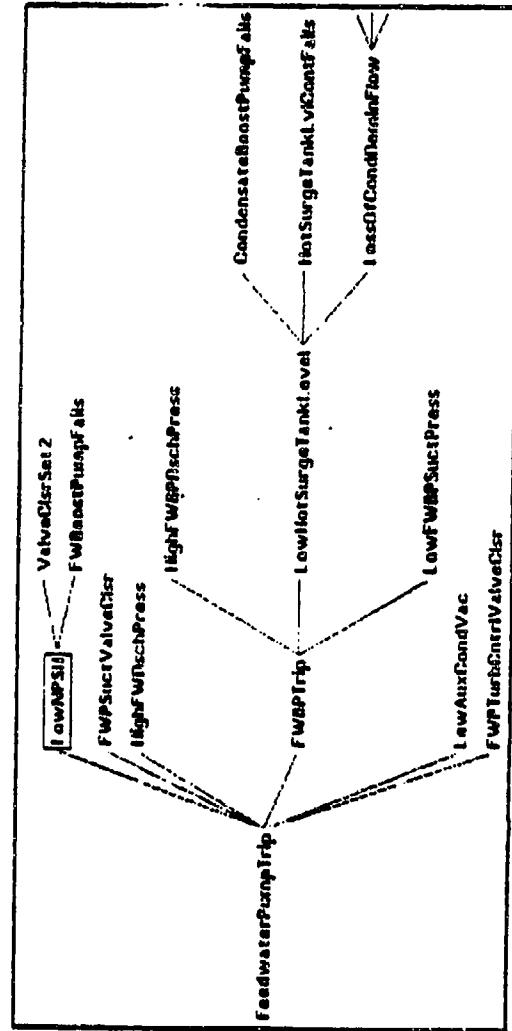


Fig. 24. Diagnostic Hierarchy below FW Pump Trip in DVS.

Hardcopy of output from PSMS and DVS

PSMS New Monitoring Cycle beginning with Safety Function Monitoring

PSMS No condition threatens any safety goal, looking at general conditions of deviations from normality

PSMS No general deviations exist, proceeding to look at other events

PSMS No condition indicates any abnormal event

PSMS There is no need to do any diagnosis

PSMS New Monitoring Cycle beginning with Safety Function Monitoring

PSMS No condition threatens any safety goal, looking at general conditions of deviations from normality

PSMS No general deviations exist, proceeding to look at other events

PSMS *****

PSMS The FeedwaterPumpTrip abnormal event is established

Message is being sent to DPMS to trigger Plan for FeedwaterPumpTripAE

PSMS Following is the message from DPMS

FeedwaterPumpTripAE-Planner triggered by DPMS

Hardcopy of output from PSMS and DVS

**PSMS Do you want to do further diagnosis for cause of
FeedwaterPumpTrip (answer y or No no N n) ?**

Y

Diagnosis for cause of FeedWaterPumpTripAE initiated

PSMS The cause of FeedWaterPumpTripAE is LowNPSH

Message is being sent to DPMS to trigger Plan for LowNPSH

PSMS Following is the message from DPMS

LowNPSH lower fault plan triggered by DPMS

PSMS The diagnosis has completed it's evaluation,

**PSMS New Monitoring Cycle beginning with Safety Function
Monitoring**

PSMS *****

PSMS The InventoryControlSG safety goal is established

**Message is being sent to DPMS to trigger Plan for
InventoryControlSG**

PSMS Following is the message from DPMS

InventoryControlSG-Planner triggered by DPMS

Hardcopy of output from DPMS

>> DPMS is ready to receive messages from PSMS

>> DPMS *** Message received from PSMS for Threat to Safety or Abnormal Event *******

>> DPMS FeedwaterPumpTripAE-Planner is initiated and is in action now

>> ** Generating operation guidance **

>> Prerequisites of FeedwaterPumpTripAE-Plan being checked

>> Prerequisites of FeedwaterPumpTripAE-Plan are available

>> Achieve goal of FeedwaterPumpTripAE-Plan being checked now

>> FeedwaterPumpTripAE-Plan is in action after the prerequisite and achieve goal checks

>> Prerequisites of ONI-N27 being checked

>> Prerequisites of ONI-N27 are available

>> Achieve goal of ONI-N27 being checked now

>> ONI-N27 is in action after the prerequisite and achieve goal checks

>> Prerequisites of ONI-N27-3.0 being checked

>> Prerequisites of ONI-N27-3.0 are available

>> Achieve goal of ONI-N27-3.0 being checked now

>> ONI-N27-3.0 is in action after the prerequisite and achieve goal checks

RECOMMENDED OPERATION IS

RUNBACK RecircFlowContValvePosA1

Did you Succeed the operation? -->

Y

Is the value of reactor recirculation FCV-A position (B33-K695A)

< 22 percent open ? -->

Y

RECOMMENDED OPERATION IS

RUNBACK RecircFlowContValvePosB1

Did you Succeed the operation? -->

Y

Is the value of Reactor recirculation FCV-B Position (B33-K695B) < 22 percent open ? -->

>> DPMS *** Message received directly from DVS for the cause of Abnormal Event FeedWaterPumpTrip**

>> DPMS LowNPSH is initiated and is in action now

***** TAKE CARE OF LOW NPSH**

Y

>> ONI-N27-3.1.1 is in action after the prerequisite and achieve goal checks

Hardcopy of output from DPMS

>> **** Generating operation guidance ****
>> **Prerequisites of RPVControlSG-Plan being checked**
>> **Prerequisites of RPVControlSG-Plan are available**
>> **Achieve goal of RPVControlSG-Plan being checked now**
>> **RPVControlSG-Plan is in action after the prerequisite and achieve goal checks**
>> **PEI-B13-3.0.1 is in action after the prerequisite and achieve goal checks**

RECOMMENDED OPERATION IS

Check if the reactor scram has initiated

Did you Succeed the operation? -->

Y

>> **Prerequisites of PEI-B13-3.1.1-MONITORA being checked**

Are all control rod positions <= 2 ? -->

Y

RECOMMENDED OPERATION IS

SET ReactorModeSwitch1 to Shutdown

Did you Succeed the operation? -->

Y

>> **RPVPowerControl is not required as its achieve goal is already true**

>> **RPVLevelControl is in action after the prerequisite and achieve goal checks**

>> **PEI-B13-3.3.1 is in action after the prerequisite and achieve goal checks**

RECOMMENDED OPERATION IS

MANUALLY-ISOLATE MSIVOutboardVlvDB

Did you Succeed the operation? -->

Y

Are all of the main steam line outboard isolation valves Isolated ? -->

Y

>> **PEI-B13-3.3.2-MONITOR is in action after the prerequisite and achieve goal checks**

RECOMMENDED OPERATION IS

MAINTAIN RCIC Turbine speed below the RCIC Turbine Speed Limit but above 2000 RPM

Did you Succeed the operation? -->

Y

>> **RPVLevelControl has succeeded**

>> **RPVPressureControl is not required as its achieve goal is already true**

>> **DPMS RPVControlSG-Plan has succeeded**

>> **DPMS RPVControlSG-Planner has succeeded**

Is the value of RPV wide range level instruments > 178 inches ?

-->

Y

>> **DPMS InventoryControlSG-Planner has succeeded**

>> **DPMS No more active plans to be pursued in NPPGoals**

Hardcopy of output from DPMS

*** Reactor power should be maintained BELOW a maximum of 100 percent by adjusting the recirculation flow

RECOMMENDED OPERATION IS

ADJUST recirculation flow

Did you Succeed the operation? -->

Y

Is the value of TOTAL REACTOR FW FLOW (C34-N002A&2B) =

TotalReactorSteamFlowDB ? -->

Y

>> ONI-N27-3.1.1 has succeeded

>> ONI-N27-3.0 has succeeded

>> Prerequisites of ONI-N27-4.0 being checked

>> Prerequisites of ONI-N27-4.0 are available

>> Achieve goal of ONI-N27-4.0 being checked now

>> DPMS ***** Message received from PSMS for Threat to Safety or Abnormal Event *****

>> DPMS InventoryControlSG-Planner is initiated and is in action now

>> ** Generating operation guidance **

>> Prerequisites of InventoryControlSG-Plan being checked

>> Prerequisites of InventoryControlSG-Plan are available

>> Achieve goal of InventoryControlSG-Plan being checked now

>> InventoryControlSG-Plan is in action after the prerequisite and achieve goal checks

*** The reactor water level has decreased BELOW the scram setpoint --- entering PEI-B13 RPVControl

>> PEI-B13 is in action after the prerequisite and achieve goal checks

>> PEI-B13-3.0.1 is in action after the prerequisite and achieve goal checks

RECOMMENDED OPERATION IS

Check if the reactor scram has initiated

Did you Succeed the operation? -->

N

>> DPMS InventoryControlProcedure has not succeeded

>> DPMS InventoryControlSG-Planner has not succeeded, recovery action started

>> DPMS It is expected that RPVControlSG-Planner will be threatened

>> DPMS RPVControlSG-Planner is initiated and is in action now

Hardcopy of output from DPMS

*** Reactor power should be maintained BELOW a maximum of 100 percent by adjusting the recirculation flow

RECOMMENDED OPERATION IS

ADJUST recirculation flow

Did you Succeed the operation? -->

Y

Is the value of TOTAL REACTOR FW FLOW (C34-N002A&2B) =

TotalReactorSteamFlowDB ? -->

Y

>> ONI-N27-3.1.1 has succeeded

>> ONI-N27-3.0 has succeeded

>> Prerequisites of ONI-N27-4.0 being checked

>> Prerequisites of ONI-N27-4.0 are available

>> Achieve goal of ONI-N27-4.0 being checked now

>> DPMS ***** Message received from PSMS for Threat to Safety or Abnormal Event *****

>> DPMS InventoryControlSG-Planner is initiated and is in action now

>> ** Generating operation guidance **

>> Prerequisites of InventoryControlSG-Plan being checked

>> Prerequisites of InventoryControlSG-Plan are available

>> Achieve goal of InventoryControlSG-Plan being checked now

>> InventoryControlSG-Plan is in action after the prerequisite and achieve goal checks

*** The reactor water level has decreased BELOW the scram setpoint --- entering PEI-B13 RPVControl

>> PEI-B13 is in action after the prerequisite and achieve goal checks

>> PEI-B13-3.0.1 is in action after the prerequisite and achieve goal checks

RECOMMENDED OPERATION IS

Check if the reactor scram has initiated

Did you Succeed the operation? -->

N

>> DPMS InventoryControlProcedure has not succeeded

>> DPMS InventoryControlSG-Planner has not succeeded, recovery action started

>> DPMS It is expected that RPVControlSG-Planner will be threatened

>> DPMS RPVControlSG-Planner is initiated and is in action now

References

1. Sharma, D.D., "A Knowledge Base Framework for Procedure Synthesis and its Application to the Emergency Response in a Nuclear Plant" (Ph.D. Dissertation, The Ohio State University, 1986)
2. Hashemi, S., "Expert Systems Application to Nuclear Power Plant Malfunction Diagnosis and Sensor Validation". (Ph.D. Dissertation, The Ohio State University, 1988)
3. Bhatnagar, R, Gandikota M.S., Davis J.F., Hajek, B.K., Miller D.W., and Stasenko J.E., "An Intelligent Database for Process Plant Expert Systems". Presented at The Instrument Society of America International Conference and Exhibit, Houston, TX, Oct. 1988.
4. Yamada, N, Chandrasekaran B. and Bhatnagar R., "A Knowledge Based System for Dynamic Procedure Management Based on Generic Task Methodology" (Unpublished Technical Report, The Ohio State University, 1987)
5. Stasenko, J.E., "The Implementation of Domain Knowledge into a Computer Based Advisory System for Nuclear Power Plant Operators". (M.S. Thesis, The Ohio State University 1988).
6. Hashemi, S., Hajek, B. K., Miller, D. W., Chandrasekaran, B., Punch III, W. F., "An Expert System for Sensor Data Validation and Malfunction Detection." presented and published in the Proceedings of the ANS Conference on Artificial Intelligence and other Innovative Computer Applications in the Nuclear Industry, Snowbird, Utah, 1987.
7. Bylander, T. and Mittal, S., "CSRL: A Language for Classificatory

- Problem Solving and Uncertainty Handling," AI Magazine 7(2):66-77, 1986.
8. Brown, D. C., and Chandrasekaran, B., "Knowledge and Control for Mechanical Design Expert System," IEEE Computer 19(7):92-100, 1986.
 9. Chandrasekaran, B., "Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks," presented and published in the Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, Italy, August, 1987.
 10. Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design," IEEE Expert, Fall, 1986, pp. 23 - 30.
 11. Sharma, D. D., Miller, D. W., and Chandrasekaran, B., "Design of an Artificial Intelligence System for Safety Function Maintenance," Transactions of the American Nuclear Society, Vol. 50, pp. 294 - 297, 1985.
 12. Corcoran, W. R., Church, J.F., Cross, M.T., and Guinn, W.N., "The Critical Safety Functions and Plant Operation," Nuclear Technology, Vol. 55, pp. 690 - 712, 1981.
 13. Meijer, C.H., "Critical Function Expert System for Nuclear Power Plants", Presented and Published in the Proceedings for a Workshop on AI Applications to Nuclear Power, Sponsored by EPRI, May 1984.
 14. Hajek B.K., Miller D.W., Bhatnagar R., Stasenko J.E., Punch W.F., and Yamada N., " A Generic Task Approach to a Real Time Nuclear Power Plant Fault Diagnosis and Advisory System". Presented at the

International Workshop on Artificial Intelligence for Industrial Applications, Hitachi City, Japan, May 1988.

15. Laffey T.J., Cox P.A., Schmidt J.L., Kao S.M., and Read J.Y., "Real-Time Knowledge Based Systems", AI Magazine 9(1): 27-45.
16. Grounwater, E.H., Donnell, M.L., and Archer, M.A., "Approaches to Verification and Validation of Expert Systems for Nuclear Power Plants", EPRI NP-8236, July, 1987.
17. Bullock, J.B., "Seminar on the Applications of On-Line Computers to Nuclear Reactors", Nuclear Safety Vol. 10 Mar-Apr, 1969.
18. Long, A.B., "Computerized Operator Aids", Nuclear Safety, Vol 25, July-August, 1984.
19. Lewis, J.R., "Safety Systems Status Monitoring", Nuclear Safety, Vol. 26, pp. 459-467, July-August, 1985.
20. Naser, J., R. Colley, J. Gaiser, T. Brockmire and S. Engle, "A Fuel Insert Shuffle Planner Expert System", presented and published in The Proceedings of Expert Systems Applications in Power Plants, EPRI, Boston, MA. 1987.
21. RTAD Real Time Analysis and Display, Nuclear Education and Training Services, Inc. Unpublished Training Manual, Columbus, Ohio.
22. Lloyd, B.A., D.K. Sharma, and C.K. Brede, "A Generator Expert Monitoring System (GEMS)", presented and published in The Proceedings of Expert Systems Applications in Power Plants, EPRI, Boston, MA. 1987.
23. Stefanini, A. and M. Gallanti "Expert System Applications in Power Generation and Distribution: A Survey on the Ongoing Projects at

- CISE", presented and published in The Proceeding of Expert Systems Applications in Power Plants, EPRI, Boston, Ma. 1987.
24. Neuschaefer, C.H., P.W. Rzasa, E. Filshtein, et.al., "Application of C-E's Generic Diagnostic System to Power Plant Diagnostics", presented and published in The Proceeding of Expert Systems Applications in Power Plants, EPRI, Boston, Ma. 1987.
 25. Berg, O., Rolf-Einar Grini, M. Yokobayashi, "Early Fault Detection and Diagnosis for Nuclear Power Plants", presented and published in The Proceeding of Expert Systems Applications in Power Plants, EPRI, Boston, Ma. 1987.
 26. Naito, N., A. Sakuma, K. Shigeno, "A Real-Time Expert System for Nuclear Power Plant Failure Diagnosis and Operational Guide", Nuclear Technology, Vol. 79, Dec. 1987.
 27. Petrick, W., K. Ng, C. Stuart and D. Cain, "A Production System for Computerized Emergency Procedures Tracking", presented and published in The Proceeding of Expert Systems Applications in Power Plants, EPRI, Boston, Ma. 1987.
 28. Touchton, R.A., A. Gunter and D. Cain, "REALM: An Expert System for Classifying Emergencies", presented and published in The Proceeding of Expert Systems Applications in Power Plants, EPRI, Boston, Ma. 1987.
 29. U.S., Nuclear Regulatory Commission, Functions and Operations of Nuclear Power Plant Crews, Oak Ridge, tennessee, USNRC Technical Report, NUREG/CR-2587, ORNL/TM-8237, April, 1982, pp. 11-28.
 30. U.S., Nuclear Regulatory Commission, TMI-2 Lessons Learned Task Force Final Report, Washington, D.C., USNRC Technical Report, NUREG-

0888, October, 1979.

31. General electric Company, Operator Training Services, Emergency Operating Procedures Fundamentals, Technical Report, January 1981.
32. Miller D.W., B.K. Hajek, R. Bhatnagar and J.E. Stasenke, " Installation and Evaluation of a Computer-Based Operator Advisor for Nuclear Power Plant Operators", DOE Research Grant Proposal, Nuclear Engineering Program, Mechanical Engineering Department, The Ohio State University, 1988.
33. Kirk D.B and A.E. Murray, "Verification of Expert Systems for Nuclear Power Plant Applications", EPRI NP 5978, August 1988.

Appendix J

What Kind of Information Processing Is Intelligence?

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
July 1987

WHAT KIND OF INFORMATION PROCESSING IS INTELLIGENCE?
A PERSPECTIVE ON AI PARADIGMS AND A PROPOSAL

B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Note: Section 3 is a minipaper on Connectionism, and can be read separately.

This paper is an extended version of talks that I have given at the AAAI Workshop on Foundations of Artificial Intelligence, Las Cruces, New Mexico, in February, 1986, and at the Workshops on Theoretical Issues in Conceptual Information Processing, Philadelphia, Pennsylvania, in August, 1986, and Washington D.C., June 1987. This paper will appear in *Source Book on the Foundations of AI*, Partridge and Wilks, Editors, Cambridge University Press.

Table of Contents

1. AI as a Science of Intelligence	1
1.1. Intelligence as Information Processing on Representations	2
2. AI Theories from the 40's to the 60's	3
2.1. Pre- and Quasi-Representational Theories	3
2.2. Early AI Work Based on Symbolic Representations	5
3. On the Nature of Representations: Connectionism vs The Symbolic Paradigm	5
3.1. The Roots of the Debate	6
3.2. Symbolic and Non-Symbolic Representations	7
3.3. Connectionism and Its Main Features	8
3.4. Is Connectionism Merely An Implementation Theory?	9
3.5. Need for Compositionality	10
3.6. Information Processing Level Abstractions	11
3.7. Learning to the Rescue?	12
3.8. The Domains for Connectionism and Symbolic Computations	13
3.9. Transition to the Rest of the Paper	16
4. Current Styles of Theory-Making in AI	16
4.1. Architectural Theories	17
4.2. Theories of Intelligence Based on Abstract Logical Characterization of Agents	20
4.3. Logic for Representation	22
4.4. Intelligence Has Other Functions Than Correctness	23
4.5. Generic Functional Theories of Intelligence	23
4.6. Generic Information Processing Strategies in Diagnostic Reasoning	25
4.7. Functional Theories: Heuristic Becomes Epistemic	27
5. A proposal on the Nature of Intelligent Information Processing	28
6. Concluding Remarks	29

July 10, 1987

What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and A Proposal

B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

1. AI as a Science of Intelligence

Theoretical and empirical work in artificial intelligence (AI) has now gone on for close to thirty years, but few minds have been changed about most of the central philosophical issues surrounding mind. People who asserted in the early days of AI that machines cannot think still assert that machines cannot think. People who believed that machines cannot be conscious or feel pain still believe that machines cannot be conscious or feel pain. Even with respect to the idea that computation over discrete symbol systems, the so-called "symbolic paradigm" (Smolensky, 1988), is a pretty good basis for capturing intelligence -- by far the dominant paradigm in AI and the reason AI is closely associated with computer science -- there are recurring doubts. In addition to the people out there who have always felt we need holograms or chemistry for a proper account of intelligence, we now have a connectionist school that rejects the symbolic paradigm for intelligence. In spite of all this, work in AI has steadily attracted its share of philosophers and psychologists -- not to speak of people who see the commercial possibilities of mechanized intelligence -- who feel that AI is exciting and important both in its overall view of intelligence as well as in some of its concrete achievements.

Minds and Intelligences. Let us make a useful distinction which might eliminate at least some of the arguments about AI: the distinction between "intelligence" and "mind." Many discussions on the philosophical implications of AI-- e.g., the numerous articles of the 50's and 60's on minds and machines-- equated the question, "Can machines be intelligent?" with "Are minds machines?". There is a useful alternative to this equation of mind and intelligence, viz., that intelligence is a *tool* of the mind. In fact there is a tradition in Hindu and Buddhist philosophies which embodies precisely such a distinction: it views intelligence as an internal sense organ much as sight is an external sense organ. As a sense organ, it interprets the world and makes the information available to the "watcher." My aim in making this distinction here is not to stake an ultimate position about the irreducibility of mind to mechanism, but merely to remove from discussion some elements about which AI as a technical discipline has nothing to say at this time. Even the most rabid mechanist within AI will need to admit that while AI may have impressively useful things to say about cognition and perception, it simply has nothing technical -- at this point -- to say about consciousness, feelings, will, etc. Thus from a technical viewpoint, I want to take intelligence, not mind, as the current subject matter of AI. If we have succeeded in taking vitalism out of our limbs and cells, we hope similarly to take mysticism at least out of intelligence.

Paradigmatic confusion in AI. In spite of what I regard as significant achievements of AI in beginning to provide a computational language to talk about the nature of intelligence, the not so well-kept secret is that AI is internally in a paradigmatic mess. There is really no broad agreement on the essential nature or formal basis of intelligence and the proper theoretical framework for it.

1.1. Intelligence as Information Processing on Representations

However, let us first seek some unities. There is something that is shared *almost universally* among workers in AI: "Significant (all?) aspects of cognition and perception are best understood modeled as *information processing activities on representations*." The dominant tradition within AI has been the symbolic paradigm¹. On the other hand, modern connectionists (and the earlier perceptron theorists) offer largely analog processes implemented by weights of connections in a network. Stronger versions of the symbolic paradigm have been proposed by Newell as the physical symbol system hypothesis (Newell, 1980) and elaborated by Pylyshyn (1984) in his thesis that computation is not simply a metaphorical language to talk about cognition, but that cognition is literally computation over symbol systems. It is important to emphasize that this thesis does not imply a belief in the practical sufficiency of current von Neuman computers for the task, or a restriction to serial computation. Often disagreements with the symbolic paradigm turn out to be arguments for parallel computers of some type rather than arguments against computations on discrete symbolic representations.

The above description of intelligence as information processing does not, however, characterize the class of intelligent processes well enough within the class of all information processing transformations. Is there something that can be recognized as the *essential* nature of intelligence that can be used to characterize all its manifestations: human, alpha-centaurian and artificial? It is possible that intelligence is merely a somewhat random collection of information processing transformations acquired over eons of evolution, but in that case there can hardly be an interesting science of it. It is also possible that there need not be anything that particularly restricts attempts to make intelligent machines, i.e., while there may well be characterizations of human intellectual processes, they need not be taken to apply to other forms of intelligence. While in some sense this seems right -- human intellectual processes do not bound the possibilities for intelligence -- nevertheless I believe that there is an internal conceptual coherence to the class of information processing activities characterizing intelligence. The oft-stated dichotomy between the simulation of human cognition versus making machines smart is a temporarily useful distinction, but its implication that we are talking about two very different phenomena is, I believe, incorrect. In any case, a task of AI as a science is to explain human intelligence. The underlying unity that we are seeking can be further characterized by asking, "What is it that unites Einstein, the man on the street in a western culture, and a tribesman in a primitive culture, as information processing agents?"

In this paper, my aim is to give a broad brush treatment of the attempts to understand the nature of intelligence. By their very nature, such broad brush accounts tend to treat history a bit too neatly. Another consequence of this is that an approach might be treated as belonging to a certain class, while the authors of the approach in question might not share the implications of the classification. But a treatment in such broad terms is nevertheless necessary to make sense of a field such as AI which is in some degree of conceptual confusion about its foundations.

¹ I am unhappy with this term to describe the commitment to computation over discrete symbolic systems, since I think that all representations are symbolic, otherwise they wouldn't be representations. There is really no satisfactory, generally agreed brief term for this. Fodor and Pylyshyn (1988) use the term "classical" models. Dennett (1986) uses the term, "High Church Computationalism" and so on. I will stick with the terms "symbolic paradigm," "symbolic approaches," and "symbolic computationalism" to refer to computation over discrete symbol systems.

My aim is not to give a quick tutorial on the history of AI, or even the various technical ideas in AI. I am only concerned with what AI has had to say about the question, "What kind of information processing is intelligence?" Also, this paper is really written for the AI researcher who already knows the various theories. What I plan to do is offer a view of how the field really works as a discipline, and how some disagreements can be understood only by tracing them to the root of the problem: disagreements about the nature of the science.

2. AI Theories from the 40's to the 60's

2.1. Pre- and Quasi-Representational Theories

Let us now trace the various streams in AI that attempted to come to grips with the nature of intelligence. The period under survey can be characterized as a transition from formalisms with an essentially non-representational character through ideas which oscillated between brain-level vs mind-level representations finally to a clear dominance of discrete symbolic representations within AI and emphasis on higher cognitive phenomena.

The earliest of the modern attempts in this direction was the *cybernetics* stream, associated with the work of Wiener (1948) who laid some of the foundations of modern feedback control². The importance of cybernetics was that it suggested that *teleology could be consistent with mechanism*. The hallmark of intelligence was said to be *adaptation*, and since cybernetics seemed to provide an answer to how this adaptation could be accounted for with feedback of *information*, and also account for teleology (e.g., "The *purpose* of the governor is to keep the steam engine speed constant"), it was a great source of early excitement for people attempting to model biological information processing. However, Cybernetics never really became the language of AI, because it did not have the richness of ontology to talk about cognition and perception: while it had the notion of information processing in some sense, i.e., it had goals and mechanisms to achieve them, it lacked the notion of computation, not to mention representations.

Modeling the brain as automata (in the sense of automata theory) was another attempt to provide a mathematical foundation for intelligence. For example, the finite automata model of *nervnets* that McCulloch and Pitts (1943) proposed was among the first concrete postulations about the brain as a computational mechanism. Automata models were computational, i.e., they had states and state transition functions, and the general theory dealt with what kinds of automata can do what kinds of things. While this was a source of great excitement -- one should try to imagine being present at the time when the computer, information theory and the automata theories were all being born at about the same time, and the sense of exhilaration that must have resulted from the thought that a formal language in which to talk about minds and brains was within reach! -- in retrospect, automata theory didn't have enough of the right kind of primitive objects for talking about the phenomena of cognition and perception. What AI needed was not theories *about* computation but theories which were descriptions of particular computations, i.e., really programs that embody theories of cognition. Naturally enough, automata theory evolved into the formal foundation for some aspects of computer science, but its role in AI *per se* tapered off.

Another strain, which was much more explicit in its commitment to seeking intelligence by

²Cybernetics as a movement had broader concerns than the issues surrounding feedback control, as applied by Wiener to understanding control and communication in animals and machines. Information and automata theories and neural nets as automata were all part of the cybernetic milieu of bringing certain biological phenomena under the rigor of formalisms. I discuss automata and neural nets shortly.

modeling its seat, the brain, looked at neurons and neural networks as the units of information processing out of which thought and intelligence can be explained and produced. Neural net simulation and the work on Perceptrons (Rosenblatt, 1962) are two major examples of this class of work. Its lineage can be traced to Hebb's work (Hebb, 1949) on cell assemblies which had a strong effect on psychological theorizing. Hebb proposed a dynamic model of how neural structures could sustain thought, how simple learning mechanisms at the neural level could be the agents of higher level learning at the level of thought. In retrospect, there were really two rather distinct kinds of aims that this line of work pursued. In one, an attempt was made to account for the information processing of neurons and collections of them. To the extent that it is generally granted that neural structures form the implementation medium of human intelligence and thought, this seems like an eminently important line of investigation. In fact, over the years, concrete identifications have been made of particular functions computed by particular neural structures in the brain, and these data may eventually form the empirical basis of any theory of how brains and minds can be bridged analytically.

In the other line of work in neural models -- prefiguring the claims of latter day connectionism -- the attempt is to explain intelligence directly in terms of neural computations. Since in AI explanation of intelligence takes the form of constructing artifacts which are intelligent, this is a tall order -- the burden of producing programs which simulate neural-like mechanisms on one hand, and at the same time do what intelligent agents do: solve problems, perceive, explain the world, speak in a natural language, etc. is a heavy one. There is a problem with the level of description here -- the terms of neural computation seem far removed from the complex content of thought -- and bridging it without hypothesizing levels of abstraction between neuronal information processing and highly symbolic forms of thought is difficult. In other words, even if it is true that the brain is made up completely of neural structures of certain types whose behavior is fully understood, and if one is given a bucketful of such neural structures one would still be not very close to constructing a natural language understanding program without theories of knowledge and syntax and semantics. The general temptation in this area has been to sidestep the difficulties by assuming that appropriate learning mechanisms at the neural level can result in sufficiently complex high level intelligence, much as it presumably occurred in evolution, so that the designer of the artifact need not have theories of cognition or perception at levels higher than the neural level. But the difficulty of getting the necessary learning to take place in less than evolutionary time has generally resulted in the neural network level not being a serious contender for AI theory making and system construction until a new generation of connectionist models began to admit representations of high level abstractions directly. Because it raises not only the level of abstraction issues but also issues about the nature of representations, I propose to discuss connectionism separately (Section 3).

A large body of work, mainly statistical in character, developed under the rubric of pattern recognition (see a text such as (Duda, 1973)). It identified the problem of recognition with classification and developed a number of statistical algorithms for classification. While it had a number of representational elements (the object in question was represented as a vector in a multidimensional space) and shared some of the concerns with the perceptron work (linear or nonlinear separability of patterns in N -dimensional spaces), it developed into a mathematical discipline of its own without making a significant impact on the overall concerns of AI. In (Chandrasekaran, 1986a), I discuss how more flexible representations are increasingly needed for even the classification problem as the complexity of the problem increases.

A number of reasons can be cited for the failure of all this class of work, viz., perceptrons, neural nets, and statistical classification to hold center stage in AI. The loss of interest in perceptrons is often attributed to the demonstration by Minsky and Papert (1969) of their inadequacies. I believe, however, that their demonstration was in fact limited to single layer perceptrons, and was not the real reason for their disappearance from the scene. The real reason, I believe, is that powerful

representational and representation manipulation tools were missing. The alternative of discrete symbolic representations quickly filled this need, and provided an experimental medium of great flexibility.

2.2. Early AI Work Based on Symbolic Representations

The final transition to discrete symbolic representations was rather quick. The mathematics of computability also made some investigations along this line attractive and productive. The end of the period saw not only a decisive shift towards representational approaches, but the particular kind of representationalism that became the common currency was the symbolic paradigm.

Early work in this computationalist spirit took on two major forms:

1. Show mathematically that certain functions thought to be characteristic of intelligence were *computable*, (e.g., induction machines of Solomonoff (1957), Gold's work on learning of grammars (Gold, 1967)). From the viewpoint of *constructing* artifacts that would perform these functions these results would in fact have been depressing if notions of *complexity of computation*, which were later to be developed in computer science with great elegance and precision, had been available at that time. The algorithms proposed were typically computationally intractable. However, in reality, this family of results was important to AI in making the idea of machine intelligence theoretically more plausible by showing that the mental functions as defined by reasonable appeal to intuition were in fact computable -- a certain amount of mysticism that would otherwise surround terms such as "induction" and "learning" was thus eliminated³.
2. Demonstrate the possibility of AI by building computer programs that solve problems requiring intelligence. Game playing programs, the Analogy program of Evans, the scene analysis program of Guzman and the Logic Theorist and the heuristic compiler of Newell and Simon, etc., etc., showed that the underlying features of intelligence of which they were meant to be demonstrations were in fact capable of artifactual embodiment. Of course the intent of these programs was not merely that -- each of which was also an exploration of a general theory of some sort, problem solving, scene analysis, analogical reasoning, etc.-- but not all of them led to more general theories about fundamental issues in AI. But, whatever their original intent, all these early programs ended up playing the roles of "realities in the field" to counteract the intellectual milieu of that time which resisted the notion of intelligence as mechanical.

In general, the net result of most (not all) of this work was socio-psychological: it made the idea of AI plausible, and blunted the first round objections, such as, "Ah, but machines cannot learn," and "Ah, but machines cannot create." These early programs were also the means by which psychologists and philosophers became aware of the new kid on their block. The attention that AI gained at that time has continued to this day.

3. On the Nature of Representations: Connectionism vs The Symbolic Paradigm

Let me restate some of the terminology here. I have called the hypothesis that intelligence can be

³It is an open question whether our understanding of induction, e.g., was in fact genuinely advanced from the viewpoint of building machines to perform them.

accounted for by algorithmic processes which interpret discrete symbol systems the *symbolic paradigm* or *symbolic approaches*. Let us call the alternative to this the *non-symbolic paradigm or approaches*, for lack of a better word. Connectionism is an example of this alternative, though not the only one.

A source of confusion is that connectionist theorists often use algorithmic language for describing parts of their 'systems' behavior. It is my belief (see Section 3.3) that such an algorithmic specification is quite irrelevant, and does not involve basic representational commitments. I want to reserve the term "symbolic" approaches to those theories which make representational commitments at the theoretical level for discrete symbol systems.

Since connectionism challenges some of the basic assumptions under which much AI work has gone on for the past several decades, it is important to spend some time examining the nature of representations, and the differences between the symbolic paradigm and connectionism in this regard. This section will be in the form of a detour, since the major part of this paper (from Sections 4 onward) will be on the theories of the past two decades in the symbolic paradigm.

3.1. The Roots of the Debate

The connectionism—symbolic computationalism debate in AI today is but the latest version of a fairly classic contention between two sets of intuitions each leading to a *weltanschauung* about how to study the phenomena of current interest. The debate can be traced at least as far back as Descartes in modern times (and to Plato if one wants to go further back) and the mind—brain dualism that goes by the name of Cartesianism. In the Cartesian worldview, the phenomena of mind are exemplified by language and thought. These phenomena may be implemented by the brain, but are seen to have a constituent structure in their own terms and can be studied abstractly. Logic and symbolic representations have often been advanced as the appropriate tools for studying these phenomena.

Functionalism in philosophy, information processing theories in psychology, and the symbolic paradigm in AI all share these assumptions. While most of the intuitions that drive this point of view arise from a study of cognitive phenomena, the thesis is often extended to include perception, as e.g. in Bruner's thesis (Bruner, 1957) that *perception is inference*. In its modern versions this viewpoint appeals to Turing's Hypothesis as providing a justification for limiting attention to symbolic computational models. These models ought to suffice, the argument goes, since even continuous functions can be computed to arbitrary precision by a Turing machine.

The opposition to this view springs from anti-Cartesian intuitions. My reading of the philosophical impulse behind anti-Cartesianism is that it is a reluctance to assign any kind of ontological independence to mind, a reluctance arising from the feeling that mind-talk is but an invitation to all kinds of further mysticisms, such as soul-talk. Thus anti-Cartesians tend to be materialists with a vengeance, and are skeptical of the separation of the mental from the brain-level phenomena. Additionally, the brain is seen to be nothing like the symbolic processor needed to support the symbolic paradigm. Instead of what is seen as the sequential and combinational perspective of the symbolic paradigm, some of the theories in this school embrace parallel, "holistic", non-symbol-processing alternatives, while others do not even subscribe to any kind of information processing or representational language in talking about mental phenomena. Those who think information processing of some type is still needed nevertheless reject processing of labeled symbols, and look to analog or continuous processes as the natural medium for modeling the relevant phenomena. In contrast to Cartesian theories, most of the concrete work in these schools deals with perceptual (or even motor) phenomena, but the framework is meant to cover complex cognitive phenomena as well. Eliminative materialism in philosophy, Gibsonian theories in psychology, connectionism in

psychology and AI, all these can be grouped as more or less sharing this perspective, even though they differ among each other in a number of issues. For example, the Gibsonian direct perception theory is anti-representational. Perception, in this view, is not inference nor a product of any kind of information processing, but is a one-step mapping from stimuli to categories of perception, made possible by the inherent properties of the perceptual architecture. All the needed distinctions are already there directly in the architecture, and no processing over representations is needed. To put it simply, the brain is all there is and it isn't a computer either.

Note that the proponents of the symbolic paradigm can be happy with the proposition that mental phenomena are implemented by the brain, which may or may not itself have a computationalist account. However, the anti-Cartesian cannot accept this duality. He is out to show the mind as epiphenomenal.

I need to caution the reader that each of these positions that I have described above is really a composite. Few people in either camp subscribe to all the features in my description of them. Most of them may not even be aware of themselves as participating in such a classic debate. In particular, many connectionists may bristle at my inclusion of them on the side of the debate that I did, since their accounts are laced with talk of "connectionist inference" and algorithms for the units. The algorithmic accounts in my view are incidental. (I discuss this further in Section 3.3.) But my account, painted with a broad brush as it is, is helpful to understand the rather diverse collection of bedfellows that connectionism has attracted.

Connectionism is a recent and less radical member of this camp. Many connectionists do not have a commitment to brain-level theory making. It is also explicitly representational, its only argument being about the medium of representation.

I believe that there is in fact a great deal of unanalyzed assumptional baggage in each of these classes of theories. I will try to show that connectionism is a corrective to some of the basic assumptions in the symbolic paradigm, but for most of the central issues of intelligence, connectionism is only marginally relevant.

As a preliminary to the discussion, I want to try to pin down, in the next subsection, some essential distinctions between the symbolic and nonsymbolic approaches to information processing.

3.2. Symbolic and Non-Symbolic Representations

Consider the problem of multiplying two integers. We are all familiar with algorithms to perform this task. We also know how the traditional slide rule can be used to do this multiplication. The multiplicands are represented by their logarithms on a linear scale, which are then "added" by being set next to each other, and the result is obtained by reading off the sum's anti-logarithm. While both the algorithmic and slide rule solutions are *representational*, in no sense can either of them be thought of as an "implementation" of the other. They make very different commitments about what is represented. There are also striking differences between them in practical terms. As the size of the multiplicands increases, the algorithmic solution suffers in the amount of time it takes to complete the solution, while the slide-rule solution suffers in the amount of *precision* it can deliver.

Let us call the algorithmic and slide-rule solutions C1 and C2. There is yet another solution C3, which is the simulation of C2 by an algorithm. C3 can simulate C2 to any desired accuracy. But C3 has radically different properties from C1 in terms of the information that it represents. C3 is closer to C2 representationally. Its symbol manipulation character is at a lower level of abstraction altogether. Given a blackbox multiplier, ascription of C1 or C2 (among others) as what is *really* going on makes for different theories about the process. Each theory makes different ontological commitments. Further, while C2 is "analog" or continuous, the existence of C3 implies that the

essential characteristic of C2 is not continuity *per se*, but a radically different sense of representation and processing than C1.

An adequate discussion of what makes a symbol in the sense used in computation over symbol systems requires a much larger space and time than I have at present (see (Pylyshyn, 1984) for a thorough and illuminating discussion of this topic), but the following points seem useful. There is a type-token distinction that seems relevant: symbols are types about which abstract rules of behavior are known and can be brought into play. This leads to symbols being labels which are "interpreted" during the process, while there are no such interpretations in the process of slide rule multiplication (except for input and output). The symbol system can thus represent *abstract forms*, while C2 above performs its addition or multiplication not by instantiating an abstract form, but by having, in some sense, all the additions and multiplications directly in its architecture.

While I keep using the word "process" to describe both C1 and C2, strictly speaking there is no process in the sense of a temporally evolving behavior in C2. The architecture directly produces the solution. This is the intuition behind the Gibsonian direct perception in contrast to the Bruner alternative of perception as inference⁴: the process of inference implies a temporal sequentiality. Connectionist theories have a temporal evolution, but at each cycle, the information process does not have a step-by-step character like algorithms do. Thus the alternatives in the non-symbolic paradigm are generally presented as "holistic."

The main point of this section is that there exists functions for which symbol and non-symbol system accounts differ fundamentally in terms of representational commitments.

3.3. Connectionism and Its Main Features

While connectionism as an AI theory comes in many different forms, they all seem share to the idea that the *representation* of information is in the form of *weights* of connections between processing units in a network, and information processing consists of (i) the units transforming their input into some output, which is then (ii) modulated by the weights of connections as inputs to other units. Connectionist theories especially emphasize a form of learning which is largely in the form of continuous functions adjusting the weights in the network. In some connectionist theories the above "pure" form is mixed with symbol manipulation processes. My description is based on the abstraction of connectionist architectures as described by Smolensky (1988). Smolensky's description captures the essential aspects of the connectionist architecture.

A few additional comments on what constitutes the essential aspects of connectionism may be useful, especially since connectionist theories come in so many forms. My description above is couched in non-algorithmic terms. In fact many connectionist theorists describe the units in their systems in terms of algorithms which map their inputs into discrete states. My view is that the discrete state description of the units' output as well as the algorithmic specification of the units' behavior is not substantially relevant. Smolensky's statement that differential equations are the appropriate language to use to describe the behavior of connectionist systems lends credence to my summary of connectionist systems.

While my description is couched in the form of continuous functions, the arguments in the Section

⁴Whether perception, if it is an inferential process, necessarily has to be continuous with cognitive processes, i.e., they all have access to one knowledge base of an agent is a completely different issue (Fodor, 1983). I am mentioning it here because the perception as inference thesis does not necessarily mean one monolithic process for all the phenomena of intelligence.

3.2 indicate that it is not in the property of continuity per se that the essential aspect of the architecture lies, but in the fact that the connectionist medium has no internal labels which are interpreted and no abstract forms which are instantiated during processing. Thus connectionist models stand in the same relationship to the symbolic models that C2 does to C1 in my discussion in Section 3.2.

There are a number of properties of such connectionist networks that are worthy of note and that explain why connectionism is viewed as an alternative paradigm to the symbolic theories.

- **Parallelism:** While theories in the symbolic paradigm are not restricted to serial algorithms, connectionist models are intrinsically parallel, in most implementations massively parallel.
- **Distributedness:** Representation of information is *distributed* over the network in a very specialized sense, i.e., the state vector of the weights in the network is the representation. The two properties of parallelism and distribution have attracted adherents who feel that human memory has a "holistic" character -- much like a hologram -- and consequently have reacted negatively to discrete symbol processing theories, since these compute the needed information from parts and their relations. Dreyfus (1979), e.g., has argued that human recognition does not proceed by combining evidence about constituent features of the pattern, but rather uses a holistic process. Thus Dreyfus looks to connectionism as vindication of his long-standing criticism of AI. Connectionism is said to perform "direct" recognition, while symbolic AI performs recognition by sequentially computing intermediate representations.
- **Softness of constraints** (Smolensky, 1988): Because of the continuous space over which the weights take values, the behavior of the network, while not necessarily unimodal, tends to be more or less smooth over the input space.

The above characteristics are especially attractive to those who believe that AI must be based more on brain-like architectures, even though within the connectionist camp there is a wide divergence about the degree to which directly modeling the brain is considered appropriate. While some of the theories explicitly attempt to produce neural-level computational structures, some others (see e.g., (Smolensky, 1988)) propose a "subsymbolic level" intermediate between symbolic and neural level theories, and yet others offer connectionism as a computational method that operates in the symbolic level representation itself. The essential idea uniting them all is that the totality of connections defines the information content, rather than representing information as a symbol structure.

3.4. *Is Connectionism Merely An Implementation Theory?*

Two kinds of arguments have been made that connectionism can at best provide possible implementations for algorithmic AI theories. The traditional one, viz., that symbolic computationalism is adequate, takes a couple of forms. In one, continuous functions are thought to be the alternative, and the fact that they can be approximated to an arbitrary degree of approximation is used to argue that one need only consider algorithmic solutions. In the other, connectionist architectures are thought to be the implementation medium for symbolic theories, much as the computer hardware is the implementation medium for software. In an Section 3.2, I have considered and rejected these arguments. I showed that in principle the symbolic and non-symbolic solutions may be alternative theories in the sense that they may make different representational commitments.

The other argument is based on a consideration of the properties of high level thought, in particular language and problem solving behavior. Connectionism by itself does not have the constructs, the argument runs, for capturing these properties, so at best it can only be a way to implement the higher level functions. I will discuss this and related points in Section 3.8.

Having granted that connectionism (actually, non-symbolic theories in general) can make a theoretical difference, I now want to argue that the difference connectionism makes is relatively small to the practice of most of AI. This is the task of the rest of Section 3.

3.5. *Need for Compositionality*

Proponents of connectionism sometimes claim that solutions in the symbolic paradigm are composed from constituents, while connectionist solutions are holistic, i.e., they cannot be explained as compositions of parts. Composition, in this argument, is taken to be intrinsically an algorithmic process.

Certainly, for some simple problems there exist connectionist solutions with this holistic character. For example, there are connectionist solutions to character recognition which directly map from pixels to characters and which cannot be explained as composing evidence about the features such as closed curves, lines and their relations. Character recognition by template matching, though not a connectionist solution, is another example whose information processing cannot be explained as feature composition. But as problems get more complex, the advantages of modularization and composition are as important for connectionist approaches as they are for house-building or algorithmic AI. A key point is that composition may be done connectionistically, i.e., it does not always require algorithmic methods.

To see this, let us consider word recognition, a problem area which has attracted significant connectionist attention (McClelland, Rumelhart, and Hinton, 1986). Let us take the word "QUEEN"⁵. A "featureless" connectionist solution similar to the one for individual characters can be imagined, but a more natural one would be one which in some sense composes the evidence about individual characters into a recognition of the word. In fact, the connectionist solution in (McClelland, et al, 1986) has a natural interpretation in these terms. The fact that the word recognition is done by composition does not mean either that each of the characters is explicitly recognized as part of the procedure, or that the evidence is added together in a step by step, temporal sequence.

Why is such a compositional solution more natural? Reusability of parts, reduction in learning complexity as well as greater robustness due to intermediate evidence are the major computational advantages of modularization. If the reader doesn't see the power of modularization for word recognition, he can consider sentence recognition and see that if one were to go directly from pixels to sentences without in some sense going through words the number of recognizers and their complexity would have to be very large even for sentences of bounded length.

To use another example, if one has a system that already recognizes "Monkey," "banana," and "Eat(a, b)", then recognizing "Monkey eats banana," without composing the constituent recognizing capabilities above would be very wasteful of resources and would require excessive learning times as well. Composition is a powerful aid against complexity whether the underlying system is connectionist or algorithmic. Of course, connectionism provides one style for composition and algorithmic methods another, each with its own "signature" in terms of the details of performance.

These examples also raise questions about the claims of distributedness of connectionist representations. For complex tasks, information is in fact localized into portions of the network. Again, in McClelland, et al's network for word recognition physically local subnets can be identified, each corresponding to one of the characters. Thus the hopes of some proponents for almost holographic distributedness of representation are bound to be unrealistic.

⁵My description of word recognition is modeled after the example given in (McClelland, et al) cited above. The word that they discuss is "TAKE".

3.6. Information Processing Level Abstractions

Marr (1982) originated the method of information processing analysis as a way of separating the essential elements of a theory from implementation level commitments. He proposed that the following methodology be adopted for this purpose. First, identify an information processing function with a clear specification about what kind of information is available for the function as input and what kind of information needs to be made available as output by the function. Then specify a particular information processing (IP) theory for achieving this function by stating what kinds of information the theory proposes need to be represented at various stages in the processing. Actual algorithms can then be proposed to carry out the IP theory. These algorithms will make additional representational commitments. For example, he specified that one of the functions of vision is to take as input image intensities in a retinal image, and produce as output a 3-dimensional shape description of the objects in the scene. His theory of how this function is achieved in the visual system is that three distinct kinds of information need to be generated: from the image intensities, a primal sketch of significant intensity changes -- a kind of edge description of the scene -- is generated, then a description of surfaces of the objects and their orientation, what he called a 2 1 2-dimensional sketch is produced from the primal sketch, and finally a 3-d shape description is generated.

Even though Marr talked the language of algorithms as the way to realize the IP theory, there is in principle no reason why portions of the implementation cannot be done connectionistically.

Thus IP level abstractions constitute the top level content of much AI theory making. In the example about recognition of the word "QUEEN" in Section 3.5, the IP level abstractions in terms of which the theory of word recognition was couched were the evidences about the presence of individual characters. The difference between schemes in the symbolic and connectionist paradigms is that these evidences are labeled symbols in the former, which permit abstract rules of compositions to be invoked and instantiated, while in the latter they are represented more directly and affect the processing without undergoing any interpretive process. Interpretation of a piece of a network as evidence about a character is a design and explanatory stance, and is not part of the actual information processing.

As connectionist structures evolve (or are built) to handle increasingly complex phenomena, they will end up having to incorporate their own versions of *modularity* and *composition*. Already we saw this in the only moderately complex word recognition example. When and if we finally have connectionist implementations solving a variety of high level cognitive problems (say natural language understanding or planning or diagnosis), the design of such systems will have an enormous amount in common with the corresponding symbolic theories. This commonness will be at the level of information processing abstractions that both classes of theories would need to embody. In fact, the content contributions of many of the nominally symbolic theories in AI are really at the level of the IP abstractions to which they make a commitment, and not to the fact that they were implemented in a symbolic structure. Symbols have often merely stood in for *abstractions* that need to be captured one way or another, and have often been used as such. The hard work of theory making in AI will always remain at the level of proposing the right IP level of abstractions, since they provide the content of the representations. The decisions about which of the transformations are best done by means of connectionist networks, and which using symbolic algorithms, can properly follow once the IP level specification of the theory has been given.

Thus, connectionist (and symbolic) approaches are both *realizations* of a more abstract level of

description, viz., the *information processing (IP) level*.⁶

Rumelhart and McClelland (1986) comment that symbolic theories that are common in AI are really explanatory approximations of a theory which is connectionist at a deeper level. To take the "QUEEN" example again, saying that the word is recognized by combining evidences about individual characters in a certain way may appear to be giving an algorithmic account, but this description is really neutral regarding whether the combination is to be done connectionistically or algorithmically. It is not that connectionist structures are the reality and symbolic accounts provide an explanation, it is that it is the IP abstractions contained in AI theories that contain a large portion of the explanatory power.

I argued, in Section 3.2, that given a function, the approaches in the symbolic and non-symbolic paradigms may make rather different representational commitments; in compositional terms, they may be composing rather different subfunctions. In this section I am arguing, seemingly paradoxically, that for complex functions the two theories converge in their representational commitments. A way to clarify this is to think of two stages in the decomposition: an architecture-independent and an architecture-dependent one. The former is an IP theory that will be realized by particular architectures for which additional decompositions will need to be made. Simple functions such as multiplication are so close to the architecture level that we only saw the differences between the representational commitments of the algorithmic and slide rule solutions. The word recognition problem is sufficiently removed from the architectural level that we saw macrosimilarities between computationalist and connectionist solutions. The final performance will of course have micro-features that are characteristic of the architecture (such as the "softness of constraints" for connectionist architectures).

Where the architecture-independent theory stops and the architecture-dependent starts does not have a clear line of demarcation. It is an empirical issue, partly related to the primitive functions that can be computed in a particular architecture. The farther away a problem is from the architectures' primitive functions, the more architecture-independent decomposition needs to be done at design time. I believe that certain kinds of retrieval and matching operations, and parameter learning by searching in local regions of space are especially appropriate primitive operations for connectionist architectures.

3.7. *Learning to the Rescue?*

What if connectionism can provide learning mechanisms such that one starts without any such abstractions represented, and the system learns to perform the task in a reasonable amount of time? In that case, connectionism can sidestep pretty much all the representational problems and dismiss them as the bane of the symbolic paradigm. The fundamental problem of complex learning is the *credit assignment problem*, i.e., the problem of deciding what part of the system is responsible for either the correct or the incorrect performance in a case, so that the learner knows how to change the structure of the system. Abstractly, the range of variation of the structure of a system can be represented as a multi-dimensional space of parameters, and the process of learning as a search process in that space for a region that corresponds to the right structure of the systems. The more complex the system, the vaster the space in which to do the search. Thus learning the correct set of parameters by search methods which do not have a powerful notion of credit assignment would work in small search spaces, but would be computationally prohibitive for realistic problems. Does connectionism have a solution to this problem?

⁶ I am indebted to Dean Allemang and Ashok Goel who noted that at this level of abstraction the distinctions between the two approaches to complex problems become small relative to their commonalities.

If one looks at particular connectionist schemes that have been proposed for some tasks such as learning tense endings (Rumelhart and McClelland, 1986b), a significant part of the abstractions needed are built into the architecture in the choice of inputs, feedback directions, allocation of sub-networks, and the semantics that underlie the choice of layers for the connectionist schemes. That is, the inputs and the initial configuration incorporate a sufficiently large part of the abstractions needed that what is left to be discovered by the learning algorithms, while nontrivial, is proportionately small. The initial configuration decomposes the search space for learning in such a way that the search problem is much smaller in size. In fact the space is sufficiently small that statistical associations can do the trick.

The recognition scheme for "QUEEN" again provides a good example for illustrating this point. In the McClelland, et al. scheme that I cited earlier essentially the decisions about which subnet is going to be largely responsible for "Q", which for "U," etc, as well as how the feedback is going to be directed are all made by the experimenter before learning starts. The underlying IP theory is that evidence about individual characters is going to be formed directly from the pixel level, but recognition of "QU" will be done by combining information about the presence of "Q" and "U," as well as their joint likelihood. The degree to which the evidence about them will be combined is determined by the learning algorithm and the examples. In setting up the initial configuration, the designer is actually programming the architecture to reflect the above IP theory of recognizing the word. An alternate theory for word recognition, say one that is more holistic than the above theory (i.e., one that learns the entire word directly from the pixels) will have a different initial configuration. (Of course, because of lack of guidance from the architecture about localizing search during learning, such a network will take a much longer time to learn the word. But that is the point: the designer recognized this and set up the configuration so that learning can occur in a reasonable time.) Thus while the connectionist scheme for word recognition still makes the useful *performance* point about connectionist architectures for problems that have been assumed to require a symbolic implementation, a significant part of the leverage still comes from the IP abstractions that the designer started out with, or have been made possible by an earlier learning phase working with highly structured configurations.

Additionally, the system that results after learning has a natural interpretation in terms of the abstractions that are needed to solve the problem; the learning process can be interpreted as having successfully searched the space for those additional abstractions that are needed to solve the problem. Thus, connectionism is one way to map from one set of abstractions to a more structured set of abstractions. Most of the representational issues remain, whether or not one adopts connectionism for such mappings.

Of course in human learning, while some of the abstractions needed are "programmed" in at various times through explicit instruction, a large amount of learning takes place without any "designer" intervention in setting up the learning structure as I described in the "QUEEN" example. But there is no reason to believe that humans start with a structure- and abstraction-free initial configuration. In fact, in order to account for the power of human learning, the initial configurations that a child starts out with will need to contain complex and intricate representations sufficient to support the learning process in a computationally efficient way.

3.8. The Domains for Connectionism and Symbolic Computations

For this discussion, a distinction between "micro" and "macro" phenomena of intelligence is useful. Rumelhart, McClelland, et al (1986) use the former term in the subtitle of their book to indicate that the connectionist theories that they are concerned with deal with the fine details of processes. A duration of 50-100 milliseconds has often been suggested as the size of the temporal "grain" for processes at the micro level. Macro phenomena take place over seconds if not minutes

in the case of a human. These evolve over time in such a way that there is a clear temporal ordering of some of its major behavioral states. For example, take the problem solving behavior represented by the GPS problem solver. The agent is seen to have a goal at a certain instant, to set up a subgoal at another instant, and so on. Within this problem solving behavior, the selection of an appropriate operator, which is typically modeled in GPS implementations as a retrieval algorithm from a Table of Connection, could be a "micro" behavior. Many of the phenomena of language and reasoning have a large macro component. Thus this domain includes, but is not restricted to, phenomena whose markings are left in consciousness as a temporal evolution of beliefs, hypotheses, goals, subgoals, etc.

Neither traditional symbolic computationalism nor radical connectionism has much use for this distinction since all the phenomena of intelligence, micro and macro, are meant to come under their particular purview. I would like to present the case for a division of responsibility between connectionism and symbolic computationalism in accounting for the phenomena of interest. Simply put, the architectures in the connectionist mold offer some elementary functions which are rather different from those assumed in the traditional symbolic paradigm. By the same token, the body of macro phenomena seems to me to have a large symbolic and algorithmic content. A proper integration of these two modes of information processing can be a source of powerful explanations of the total range of the phenomena of intelligence.

I am assuming it as a given that much of high level thought has a symbolic content to it (see (Pylyshyn, 1984) for arguments that make this conclusion inescapable). *How much* of language and other aspects of thought require this can be matter of debate, but certainly logical reasoning should provide at least one example of such behavior. I am aware that a number of philosophical hurdles stand in the way of asserting the symbolic content of conscious thought. If one is a radical behaviorist or a non-representationalist, I can see that no advantage accrues from granting that the corpus of thought, including language and reasoning, has a symbolic structure. Saying that all that passes between people when they converse is airpressure exchanges on the eardrum has its charms, but I will forego them in this discussion.

Asserting the symbolic content of macro phenomena is not the same as asserting that the internal language and representation of the processor that generates them has to be in the same formal system as that of its external behavior. The traditional symbolic paradigm has made this assumption as a working hypothesis, which connectionism challenges. Even if this challenge is granted there is still the problem of figuring out how to get the macro behavior out of the connectionist structure.

Fodor and Pylyshyn (1987) have argued that much of thought has the properties of *productivity* and *systematicity*. Productivity refers to a potentially unbounded recursive combination of thought that is possible in human intelligence. Systematicity refers to the capability of combining thoughts in ways that require abstract representation of underlying forms. Connectionism, according to Fodor and Pylyshyn, may provide some of the architectural primitives for performing parts of what is needed to achieve these characteristics, but cannot be an adequate account in its own terms. We need computations over symbol systems, their capacity for abstract forms and algorithms, to realize these properties.

In order to account for the highly symbolic content of conscious thought and to place connectionism in a proper relation to it, Smolensky (1988) proposes that connectionism operates a lower level than the symbolic, a level he calls *subsymbolic*. He also posits the existence of a *conscious processor* and an *intuitive processor*. The connectionist proposals are meant to apply directly to the latter. The conscious processor may have algorithmic properties, according to Smolensky, but still a very large part of the information processing activities that have been traditionally attributed to algorithmic architectures really belong in the intuitive processor.

A complete connectionist account in my view needs to account for how a sub- or nonsymbolic structure integrates smoothly with a higher level process that is heavily symbolic. There is the additional problem that an integrated theory has to face. Thought could be epiphenomenal. However, we know that the phenomena of consciousness have a causal interaction with the behavior of the intuitive processor. What we consciously learn and discuss and think affects our unconscious behavior slowly but surely, and vice versa. What is conscious and willful today becomes unconscious tomorrow. All this raises a more complex constraint for connectionism: it now needs to provide some sort of continuity of representation and process so that this interaction can take place smoothly.

Connectionist and symbolic computationalist phenomena, in my view, have different but overlapping domains. The basic functions that the connectionist architecture delivers are of a very different kind than have been assumed so far in AI, and thus computationalist theories need to take this into account in their formulations. A number of investigators in AI who do theories at this higher level correctly feel the attraction of connectionist style theories for some parts of their theory making. I have acknowledged the power of the connectionist claims that for some information processing phenomena, there exist nonalgorithmic schemes which make fewer (and different) commitments in terms of representational content. Where the impact of connectionism is being felt is in identifying some of the component processes of overall algorithmic theories as places where a connectionist account seems to accord better with intuitions. As I said earlier, retrieval and matching operations and low level parameter learning are places where I would think the higher level theories may choose connectionist alternatives if the fine points of performance are of theoretical importance. But, even here one should be careful about putting too much faith in connectionist mechanisms per se. As I have said several times in this section, the power for even these operations is going to come from appropriate encodings that get represented connectionistically. Thus, while memory retrieval may have interesting connectionist components to it, the basic problem will still remain the principles by which episodes are indexed and stored, except that now one might be open to these encodings being represented connectionistically. For example, I am in complete sympathy with the suggestion by Rumelhart, Smolensky, McClelland and Hinton (1986) that a schema or a frame is not explicitly represented as such, but is constructed as needed from more general connectionist representations. This does not mean to me that schema theory is only a macro approximation. Schema, in the sense of being IP abstractions needed for certain macro phenomena, is a legitimate conceptual construct, for encoding which connectionist architectures offer a particularly interesting way.

With regard to general AI and connectionism's impact on it, I would like to say, as H. L. Mencken is alleged to have said in a different context, "There is something to what you say, but not much." Much of AI (except where microphenomena dominate and computationalist AI is simply too hard edged in its performance) will and should remain largely unaffected by connectionism. I have given two reasons for this. One is that most of the work is in coming up with the information processing theory of a phenomenon in the first place. The more complex the task is the more common are the representational issues between connectionism and the symbolic paradigm. The second reason is that none of the connectionist arguments or empirical results show that the symbolic, algorithmic character of thought is either a mistaken hypothesis, purely epiphenomenal or simply irrelevant.

My arguments for and against connectionist notions in this section are not really specific to connectionist architectures that have been proposed. The arguments apply to alternatives in the nonsymbolic paradigm. e.g., all sorts of analog computers. Connectionist style architectures, especially those that deny modeling the brain level, often seem to have an air of arbitrariness about them, since it is then not clear what the constraints are: why that rather than something else? But, in fairness, these architectures ought to be viewed as exploratory, and in that sense they are contributing to our understanding of the capabilities and limitations of alternatives to the symbolic paradigm.

It seems to me that we need to find a *modus vivendi* between three significant insights about mental architectures:

- (i) A large part of the relevant content theory in AI has to do with the *what* of mental representations. I have called them IP abstractions.
- (ii) Whatever one's position on the nature of representations below conscious processes, it is clear that processes at or close to that level are intimately connected to language and knowledge, and thus have a large discrete symbolic content.
- (iii) The connectionist ideas on representation suggest how nonsymbolic representations and processes may provide the medium in which thought resides.

3.9. Transition to the Rest of the Paper

Connectionism has been important to my discussion not because its technical accomplishments have thrown down a challenge to symbolic approaches to AI, but because it replays one side of a long-running debate about the nature of the relationship between mind and brain. In spite the hopes of some of its supporters in philosophy, connectionism will not banish mind-talk, which is essentially representational and symbolic.

I now want to pick up the main thread of this paper. I want to review AI theories of the last two decades and see if a view of what makes intelligence a coherent computational phenomenon can be constructed. In view of the arguments in this section, my discussion will deal with the macro phenomena of intelligence.

In much of what follows, I will be talking within the symbolic paradigm for the reasons that I have described in this section.

4. Current Styles of Theory-Making in AI

From the viewpoint of the paradigms of intelligence that characterize the current work in AI, at the end of the first decade the computationalist paradigm emerged as the preferred one for much theory making. I see the research in this paradigm in the next two decades until the contemporary period as belonging to one of three broad groups of stances towards what a computational account of intelligence should look like. This characterization I am about to give is a personal one, and not (yet) part of the field's own self-consciousness; that is, it is really in the form of a thesis of what has been going on in the field from the perspective of a science of intelligence, and where people have been looking for answers, and what sorts of (often unconscious) assumptions about the nature of intelligence are implicit in these theories. Another caveat is that these theories are not mutually exclusive, (i.e., some important ideas appear in more than one approach, but with a flavor relevant to the approach), but constitute different ways to talk about the stuff of intelligence, and different answers to its nature. They are:

- I. *Architectural theories*
- II. *Abstract logical characterization of an agent's knowledge, and inference mechanisms that operate on this representation*
- III. *Theories that emphasize generic functional processes in intelligence. Each of these processes generates efficient inferences for a type of information processing task, and gives importance to organizational issues as a major source of this efficiency.*

In the next several sections, I consider each class of theories and examine their assumptions about the nature of intelligence.

4.1. Architectural Theories

Architectural theories, of which the Production System theory (Newell, 1973) is the most prominent, are a result of a search for a level of a machine at which *intelligence qua intelligence emerges*. In this section I will argue that such theories fail to relate the architecture to intelligence in such a way that one can see the essential role played by the architecture in the emergence of intelligence. Their Turing universality often muddles the issue. They tend to foster a search for solutions at the architecture level, when a more problem-specific solution would be more appropriate. These architectural level solutions often have a "syntactic" feel to them in comparison with more content-driven solutions at the higher level.

In architectural theories the solution to the problem of intelligence is to provide a computational architecture which is intrinsically seen as the source of intelligence. Below that level of abstraction are presumed to lie matters of implementation out of non-intelligent processes, whereas above that level, i.e., the *content* of such architectures, are agent-specific particulars of the world, the domain, etc., not of intrinsic interest to a theoretician of intelligence, because the architecture is supposed to provide *all* the necessary elements of intelligence. In particular, it is very important for these kinds of theories that the architecture being proposed be a *unitary* architecture. By unitary, I mean one level rather than multiple levels of architecture, each of which contributing some aspect of intelligence.

Let me be concrete, so that it may be clearer what I mean to include in this classification. Typically, intelligence as a whole or a large class of problems within intelligence is treated as solving and representing the information processing activity in some very general scheme: production rules, logical clauses, frames, semantic networks, or whatever. Then a basic class of inference or information processing activity on these representations is defined: forward or backward chaining in production systems, truth maintenance (Doyle, 1979) or resolution processes for logical sentences, various kinds of interpreters for frames or semantic networks, and so on. Then any particular problem, say diagnosis or design, is solved by attempting to program a solution using the underlying architecture. For example, diagnosis would be treated as particular example of truth maintenance, backward chaining, propagation of values in a network, or whatever the processes the underlying architectures directly support. Thus the diagnostic problem is reduced to programming in a given architecture. I will frequently use the production system architecture as an example to make our points regarding architectural theories. The comments are meant to apply to all unitary architecture theories.

In production systems, intelligence is viewed as a rule processor. What the production systems are actually implemented on -- say Lisp machines in AI or neural structures in natural intelligence -- are themselves unimportant from the viewpoint of intelligence, though they may be relevant from considerations of engineering issues such as speed. Similarly, in this perspective, what particular production systems contain, e.g. rules about liver diseases or about Vax configurations, is *per se* not a subject matter of the science of intelligence. The inference mechanisms at the level of the architecture -- the various chaining schemes and conflict resolution -- are however the subject matter of this science, since those mechanisms are intrinsically and intimately related to the architecture.

You can substitute your favorite architectural theory of intelligence -- frame systems, Logic architectures, belief networks, etc. -- for the production system example above, and come up with a similar analysis. The point is that the theories propose a unitary architecture as a privileged level

to capture intelligence. Whenever anyone says, "The mind is a, " you know that person is proposing an architectural solution to AI. Now, among these architectures, some of them may have more psychological evidence than others, but the problems associated with seeking the seat of intelligence at one level of the architecture.

It is true that each of these architectures has been used to build impressive systems that perform some task requiring intelligence. However, since these architectures are Turing-universal⁷, i.e., any algorithm can be implemented as a program for this architecture, it is often hard to know if the architecture *per se* is performing important work. It may merely describe an appropriate implementation. The architecture, *a priori*, does not distinguish tractable solutions from intractable ones. It does not identify good vs bad ways of organizing knowledge. Also, because it is a unitary architecture, it necessarily omits as constructs of the architecture important, higher level information-processing distinctions that are needed to give an adequate functional description of intelligence and *which may require architectural support of their own*. A well-known proponent of rule-based architectures said during a lecture, "Common sense is just millions and millions of rules." One might well respond, "Yes, and *War and Peace* is just thousands and thousands of sentences."

The case of metarules in production systems can be used to illustrate this point. In knowledge-based systems work, the rule architecture was originally offered as advantageous because it permitted a knowledge-base of domain facts, and an inference engine which implemented the control of reasoning using the facts in the knowledge base. It turned out that the control at the rule level was inadequate to perform a wide variety of tasks, so a supposedly domain-specific knowledge base increasingly acquired rules whose role was to provide the requisite control behavior and focus in problem solving. That is, it was seen that *there were phenomena essential to intelligent behavior that were above the rule level of architecture, but were not merely a collection of agent-specific world facts*. At this point, the idea of *metarules* was introduced, where each metarule was a domain-independent⁸ control rule that helped organize the knowledge base for a particular type of problem solving activity. However, while the metarule was a way of using the rule architecture to implement a control strategy in rule architectures, there was no obvious reason why the underlying lesson, viz., that we need to organize the control and knowledge base so that the facts in it were used in a certain way for certain kinds of tasks, was limited to the rule architecture. Rather the need for control knowledge in addition to domain knowledge is a lesson that can be applied to any unitary architecture. The control vs domain knowledge distinction is a high level statement about the content or role of knowledge and not a syntactic statement about a particular architecture. If the desired problem solving behavior did come about as the result of the metarule, the rule architecture was not, *a priori*, responsible; instead, the credit should go to the particular control knowledge represented by the meta rule. Thus the commitment to the unitary rule architecture at first suppressed the higher level distinctions at the level of the control strategies implicit in the different metarules. While metarules had the germ of the idea that interesting control issues were being given short shrift in rule-based approaches, it is interesting that until Clancey's work (Clancey,

⁷There is nothing in this argument that turns on whether or not intelligence can be accounted for by Turing-computable functions only.

⁸The word "domain" is a possible source of confusion. In AI, the term has come to refer almost exclusively to what one might call specific subject matter-- or a collection of facts about a miniworld -- such as medicine, whereas in many philosophical discussions about mind, I have seen the term refer to any generic faculty such as visual perception or even tasks such as natural language understanding or diagnostic reasoning. In this paper, I will use the term as it has become familiar in AI. The assumption is that no theory of intelligence need specifically deal with medical facts *per se*, but it will need to deal with phenomena such as diagnostic problem solving or natural language understanding.

1984), the syntactic aspect of metarules, not their content, dominated the discussion in rule-based system discussions. Thus much of the discussion on metarules emphasized their syntactic properties, i.e., were a contribution to rule programming. This concern with converting higher level content phenomena into syntactic solutions at the architecture level is what I mean by suppressing higher level distinctions. Another example, with somewhat serious consequences in my view, is also from the rule-based architecture example. One of the problems that arises in this architecture when the knowledge base is very large is that a large number of rules "match" a given situation, with conflicting actions proposed by each of the rules. The processor has to choose one of the rules and pursue the consequences in terms of the inferences that follow. A new problem called the "conflict resolution" problem was formulated, and a family of essentially syntactic resolution strategies were proposed, such as "Choose that rule which has more left hand side terms matching," "...has more goals on the right hand side", or "...has a higher certainty factor," etc. In truth, I claim that conflicts of this type are artifacts of the architecture, and there are higher level organizational phenomena that actually constrain only a portion of the knowledge base to be active in the first place, thus obviating the need for syntactic conflict resolution strategies.

While I have used rule architectures as examples in my discussion above, the points made are more general, and apply to unitary architectures in general. Retrieval theories based on semantic networks tend to explain differential retrieval by positing that they are due to distances measured in the number of links rather than the types of links and the knowledge embedded in the nodes and links. These unitary architectures encourage theory making at the wrong level of abstraction.

The end of this section is probably a good place to make a number of clarifications.

- The above arguments are not directed against the existence of appropriate architectures on which to implement intelligent information processing systems. In principle, it is even possible to hold that all higher level phenomena of importance can be implemented on the architecture at one particular level in such a way as to give some significant performance or construction advantages. What is being argued is that *seeking a unitary architecture as the single answer to what makes intelligence* is fraught with the problems that I have described, and misstates the nature of intelligence. I doubt that there is one level which can be identified with the emergence of intelligence. On the other hand, the discussion about whether rule architectures or alternate architectures properly capture the human information processor at one level of abstraction is not vacuous: it is a relevant and useful question. It is also important to state that researchers who have been investigating these architectural questions, e.g., Newell and Simon in the case of rule architectures, do not themselves necessarily reduce all higher level issues in intelligence to syntactic aspects of their particular architectures. It is an empirical fact, however, that these architectures do foster a unidimensional view of intelligence, resulting in important higher level questions being ignored or treated as mere programming issues at the architecture level as in the example of metarules that I discussed.
- I mentioned frame architectures as another example of unitary architectures. This is not an argument against "frames" as a functionally useful construct for intelligence; in fact, in a Section 4.5, I will offer the frame theory as an example of the right kind of theory-making in AI. In all of the discussions in this section, my argument has been against a style of theory-making which converts such a theory into a universal architecture.
- The arguments in this section are not meant to be against the *idea* of architectures to support intelligent computation. In Section 4.5 I argue that intelligent information processing comes as a variety of functional types, each of which is supported by a local "mini" architecture. Form follows function, in this as in many other designs, natural or artificial. Thus the suggestion is that instead of fixing on one form (even if true at

some level), it is more productive to identify functions and then look for forms to support them.

4.2. Theories of Intelligence Based on Abstract Logical Characterization of Agents

In many circles some version of logic is thought to be the proper language of characterizing all computation, and by extension, intelligence. By logic is meant a variant of first order predicate calculus or at least a system where the notion of truth-based semantics is central and inference-making is characterized by truth-preserving transformations.⁹ The way logic has actually been used in AI however combines a number of distinct roles that logic can play or can be claimed for it. We can begin by noting two broad roles for it:

1. logic for abstractly characterizing and analyzing what an agent knows, *vs*
2. logic as a representation for knowledge, and logical deduction as the basic information processing activity in intelligence.

First of all, there seems to be a general tendency, even among those who do not adopt the logic paradigm for knowledge representation and inference, to concede to logic the status as the appropriate language for the abstract characterization of an agent, i.e., for meta-AI analysis. While this seems like one good possibility, this does not seem to me the only or even a compelling possibility. Standing outside an intelligent agent, one can take two distinct abstract stances toward it: the agent as a performer of functions, or the agent as a knower of propositions. There are AI proposals and work that correspond to both these viewpoints. In Marr's work on vision (see Section 3.6), e.g., the agent is characterized *functionally*, i.e., by an information processing task that transforms information of one kind into that of another kind. On the other hand, the tradition in logic-based AI is one of attributing a body of knowledge to the agent. It is certainly not obvious why the knowledge view should necessarily dominate even at the level of abstract characterization. I will hold later in this paper that the functional view has superior capabilities for abstract specification of an intelligent agent.

The idea of abstract characterization of an intelligent agent through logic was first detailed by McCarthy and Hayes (1969) where they proposed the now-famous *epistemic-heuristic* decomposition of an actual intelligent agent. This distinction has echoes of the Chomskian competence/performance distinction in language. (See (Gomez and Chandrasekaran, 1981).) The agent as a knower is characterized by the epistemic component. What kinds of knowledge are to go into the epistemic component is not clear, but one would think that would depend on the theorist's view of what kinds of knowledge *characterize* intelligence. Thus it must represent a theory of the ontology of the mental stuff. The heuristic part is that part of the agent which actually makes him an efficient information processor, using the knowledge in the epistemic part to solve problems and do whatever intelligent agents do. An analogy would be that a calculator's epistemic part would be the axioms of number theory, while its heuristic part would be the particular representations and algorithms. This example also makes clear the relationship of the epistemic/heuristic distinction to the competence/performance distinction of Chomsky. McCarthy and Hayes proposed that the epistemic part be represented by a logical calculus, and in fact discussed the kind of logic and the kinds of predicates that would be needed for adequacy of representation as they conceived the epistemic part. In this attempt to separate the *what* of intelligence with the *how* of implementation, the McCarthy-

⁹ I am sure this characterization is not accurate in details. I am told that there things such as "imperative logics" where some of the above characterization might not hold good. Within AI, nonmonotonic logic relaxes the truth-preserving requirement in transformations. I believe that the thrust of my arguments nevertheless remain valid.

Hayes proposal follows a more general idea in computer science, but identifies the *what* with the propositional knowledge of the agent. This is not, however, neutral with respect to consequences.

Now it is interesting to note that the epistemic/heuristic distinction as a means of separating the essential content of an agent from the implementation details and "mere" efficiency considerations is independent of logic as a representation. As just mentioned, this kind of separation is a strong tradition in computer science, at least in proposals about software engineering, if not in practice. All that the epistemic/heuristic distinction demands is that the essential characterization of an information processor be kept separate from the implementation details. Logical representation of the epistemic part is only one alternative for doing this. As I mentioned, within AI, an alternative to McCarthy and Hayes' suggestion is Marr's proposal that this distinction be carried out by separating the information processing task (the input-output specification), the algorithm carrying out the task, and the mechanism by which the algorithm is implemented.

Even more important is that there is no self-evident way of deciding, in a theory-neutral fashion, exactly what is epistemic, and what is merely heuristic. One person might dismiss some aspect of an intelligent agent as merely heuristic (e.g., search control knowledge which helps in efficiency), while another person's theory might hold that that is precisely where the secret of intelligence lies, and thus would make it part of the epistemic component. The epistemic/heuristic distinction does not force one into an agreement about what entities ought to end up in which component. In fact, in some logical theories in AI, some important control phenomena have been moving from the heuristic component to the epistemic component, as a consensus builds in AI that a certain phenomenon is *really* not simply heuristic, but part of the stuff of intelligence. A good example of this is the development within the logic camp of families of *default or nonmonotonic logics*. It is interesting to trace the history of default logics within the logic paradigm in AI.

Now, Minsky's paper on frames (Minsky, 1975) argued against the "atomic" stand about knowledge that logic-based AI theories took, and claimed that chunking of knowledge into "frames" had a number of useful properties. In particular, if frames could stand for stereotypes of concepts, it was possible to do a form of default reasoning, where in the absence of specific information to the contrary, a reasoning system would assume default values associated with the features of a frame, thus allowing missing information to be plausibly inferred, greatly decreasing storage requirements (only non-default values, i.e., exceptions, need to be stored), and increasing retrieval speed. At first blush, all these useful properties were "heuristic" aspects, i.e., how to get the computation done faster, and thus one would think not of intrinsic epistemic interest. However, when the consensus started to develop that this form of default reasoning was one of the essential aspects of being intelligent, defaults became part of the epistemic component of AI. That is, theorists started hypothesizing the existence of something called "default reasoning" or "nonmonotonic logic" in order to account for this phenomenon in a rigorous way. Similarly, as builders of AI systems discover the utility of organizational constructs such as "plans" (which are abstract, partial solutions to problems, stored by the agent, typically indexed by the goals they help achieve), one finds that new epistemic theories such as *plan logics* get proposed and investigated. It is almost as if the lowly "heuristic" component is in fact what the action in AI is often about, while the epistemic part appropriates the nuggets of organizational wisdom that research in the heuristic component identifies.

Now of course there is nothing wrong with this as a way of making scientific progress: Let the system builders discover phenomena experimentally and let the theoreticians follow up with formalization. However, the eventual success of this kind of formalization is questionable. Is there in fact a set of inference rules that compactly characterize all and only default inferences? I propose that terms such as "nonmonotonic logic" are reifications: a complete account would require specification of a much larger set of rules than what is normally thought of as inference rules; so large as to be virtually coextensive with the entire set of distinct types of functions in which frames get used: scripts, plans, etc.

The distinct notions of abstractly characterizing an intelligent agent, via., the epistemic/heuristic distinction, and logic as a *representation* for the epistemic component, are often conflated. I just commented on how uncertain the actual allocation of information to the epistemic and heuristic components can be. Now I need to make some remarks on logic as a representation for the epistemic component.

4.3. Logic for Representation

The proposal to use logic for representation of knowledge could be in the service of two rather different purposes: one, in order to reason *about* the agent and two, to model the reasoning *of* the agent. The former is in the spirit of certain ideas in computer science where a program may be in any appropriate language, but reasoning about the program, e.g., to establish its correctness, is often done using logic. However, in practice, almost all use of logic as knowledge representation in AI has been in the service of the latter, i.e., to actually create reasoning agents.

Logic as knowledge representation makes a serious commitment to knowledge as propositions, and to True/False judgements as the basic use of knowledge. It is also closely connected to the belief that the aim of intelligence is to draw *correct* conclusions. In this view, what human beings often do, e.g., draw plausible, useful, but strictly speaking logically incorrect conclusions, is interesting as psychology, but that only shows up humans as approximations to the ideal intelligent agent, whose aim is to be *correct*.

A little digression about the nature of the ideal intelligence may be appropriate here. I believe that there has been a problem in AI due to two different senses of the word "intelligence." There is the technical sense of intelligence as the information processing activities engaged in by, possibly among others, human beings, and which is the goal of AI to understand and capture. There is another sense in which intelligent really refers to "very intelligent," i.e., some one who has been especially impressive in his or her cerebration. At least in the current western milieu, this latter quality would be denied to someone who, after all the work of thinking, was not correct in the conclusions that were drawn. Ever since late 19th century, when the foundations of mathematics showed cracks and there were considerable worries about how to be sure if conclusions were correct, which in turn pushed symbolic logic to its current rich technical accomplishments, logical reasoning has been equated with the real test of thought, *vide* the title of Boole's book, *Laws of Thought*. In addition, the content of consciousness seems to include a series of propositions, some of them beliefs, and at least for a certain kind of theorist, it seems entirely natural to model thought itself as basically manipulation of propositions and generation of new ones. In this view of thought, stream of consciousness imaginings, half-formed ideas, vague sensations at the back of the mind, how a certain idea suddenly came to occupy consciousness from the depths of the mind, etc., etc., do not count as serious subject of study from the viewpoint of intelligence as such. Hence the almost unconscious equation of thought with logical thought, and the natural attempt to seek in logic the language of representation and construction of the idealized agent.

Now, is "truth" in fact the right kind of basic interpretive framework for knowledge? Or are notions of functional adequacy, i.e., knowledge that helps to get certain kinds of tasks done, or the related notions of plausibility, relevance, etc. more effective in capturing the way agents in fact use knowledge? My 16-month old daughter, when shown a pear, said, "apple!" Is it more than mere parental pride that makes me attribute a certain measure of intelligence to that remark, when, viewed strictly as an utterer of propositions, she told an untruth? What kind of a theory of intelligence can explain that her conclusion was adequate for the occasion: she could get away with that error for most purposes -- she could eat the pear and get nourishment, e.g. -- while an equally false proposition, "It's a chair," would not give her similar advantages?

A number of theoretical advantages have been claimed for logic in AI, including *precision* and the *existence of a semantics*. The problem is that the semantics are not at the most appropriate level for the problem at hand, and logic is neither a unique nor a privileged way to be precise.

4.4. *Intelligence Has Other Functions Than Correctness*

Laws of justification are not identical to laws of thought, Boole notwithstanding. While it would be useful for an intelligent agent to have the former laws and apply them appropriately, those laws alone cannot account for the power of intelligence as a process. It seems highly plausible to me that much of the power of intelligence arises not in its ability to lead to correct conclusions, but in its ability to direct explorations, retrieve plausible ideas, and focus the more computationally expensive justification processes where they are absolutely required. Thus the power of intelligence really resides in what has been called the heuristic part, and theories of intelligence will need to be theories of that part of the decomposition, the part that is most concerned with computational feasibility. This is why organizational theories such as the frame theory, planning, and theories of memory, find their important ideas migrating to the epistemic side, which by definition, in the logic framework, is supposed to worry about the real essence of intelligence. What is interesting is that the pressure of discoveries in the heuristic side comes from efforts to actually *construct* intelligent artifacts. To the extent that explanation of intelligence as a computational phenomenon is treated within AI as the capability to construct intelligent artifacts, it is significant that it is this so-called heuristic side that has been the source of important discoveries about how the intelligent information processing can be controlled. Abstraction in the manner proposed by those who advocate logic *separates knowledge from its function*, and this leads to missing important aspects of the form and content of knowledge.

It is often argued that the epistemic/heuristic distinction is tactical: get the terms needed right, before worrying about how to actually use them in reasoning. For example, before building, say, common sense reasoners, let us get all ontology of common sense reasoning right: "know," "cause," etc. The thrust of my argument is that, as a rule, a use-independent study of such terms is likely to make distinctions that are not needed by the processing part and miss some that are.

This is not to say that logic, as a set of ideas about *justification*, is not important to intelligence. How intelligent agents discover justifications and how they integrate them with discovery procedures for a final answer that is plausibly correct, and how this done in such a way that the total computational process is controlled in complexity are indeed questions for AI as an information processing theory of intelligence. In this view logic is a brick intelligence builds, rather than the brick out of which intelligence is built.

The laudable goal of separating the knowledge necessary for intelligence from the implementation details needs in my opinion to be achieved by concentrating on the functional characteristics of intelligence. This brings me to the third set of theories.

4.5. *Generic Functional Theories of Intelligence*

The theories that I discuss in this section identify a generic, functional property of intelligence which is used to solve a "natural kind" of cognitive problem. Examples of such theories are: the GPS means-ends theory of problem solving (Newell and Simon, 1972), the Frame theory of knowledge organisation (Minsky, 1975), Schank's CD, Script (Schank and Abelson, 1977) and memory theories (Schank, 1982), our own work at Ohio State on generic types of problem solving (Chandrasekaran, 1983; 1986; 1987). These theories typically emphasize some *organizational* aspect, which facilitates some particular *class of inferences*, or *computations*, or *constructions* in a computationally efficient way. An abstract description of these processes would be replete with terms that

carry an information processing strategy connotation, such as *default*, *goals*, *subgoals*, *expectations*, *plans*, and *classification*. Knowledge of an agent is encoded using such terms. Each of these processes -- the inference mechanisms along with the knowledge structures -- constitute what we call a generic information processing strategy. Each captures a functional unit of intelligence as a process, and is generic in the sense that it is domain-independent.¹¹

In this section, I briefly discuss some of the well-known theories of this genre. In particular, I consider such strategies in knowledge-based problem solving.

Let me recapitulate some of the theories that I mentioned at the beginning of this section.

- The General Problem Solver. GPS says that a generic process available to intelligence is to treat problems in a goal/subgoal manner. In this way, the ends, i.e., the goals, of the problem solver, are matched to the means, the operators available. The agent has to have knowledge organized in a certain way, which gives information about relationship between goals, operators, etc., and a particular inference process (or control regime) called the *means-ends algorithm* is needed to use this knowledge to solve the problem. Note that the means-end method is not implicit in the problem statement, i.e., a purely logical analysis merely would specify a problem space in which the solution would lie. On the other hand, means-ends is not a particular solution algorithm for a particular problem. It is a generic information processing strategy that can be used provided that knowledge is available in a certain form.
- The frame theory. It proposes that the generic functional properties of "chunking" and "stereotyping" are important phenomena in knowledge organization. These features of the organization enable an important type of inference process called *default reasoning* to take place by using the default expectations implicit in the idea of stereotypes. Chunking and stereotyping are computationally efficient mechanisms, both for memory and for processing.
- Schank's *conceptual dependencies* and *script* theories. These are in the spirit of frame theory, but they propose contentful additions that are appropriate for certain classes of problems. CD's are particular kinds of chunks that represent action stereotypes, and enable default inferences about actions associated with verbs to be made efficiently during natural language understanding. Scripts are frames representing stereotypical action sequences, which enable expectations to mediate understanding. Both these notions (and other similar constructs in Schank's theories of memory organization) use expectations arising from stereotypes of phenomena as efficient means of computing some information about the situation.
- Plans: these are compiled, abstract, partial solutions to problems, indexed by goals. Agents which use them cut down the search for solutions enormously, again a "heuristic" advantage, but mirrored in knowledge so deeply that it becomes epistemic in character. See (Miller, et al, 1960).
- Memory Organization & Retrieval: Agents index, store and retrieve relevant events of their experience in such a way that they provide, among other capabilities, a method of reasoning (by analogy with past events, e.g.) The work of Schank's group proposes a number of types of memory chunks and indexing schemes.

¹¹Please see the earlier footnote on the word "domain" here.

- **Generic Tasks:** In our own work on knowledge-based reasoning, we have proposed a number of information processing strategies, each of which is characterized by knowledge represented using strategy-specific primitives, and organized in a specific manner. Each of the strategies also employs a characteristic inference procedure which is appropriate to the task. By showing how these strategies help in solving a computationally complex problem such as diagnosis, I hope to suggest how strategies of this kind characterize intelligence. The role of a specific set of generic strategies in diagnosis is the subject of the Section 4.6.

4.6. *Generic Information Processing Strategies in Diagnostic Reasoning*

Formally, the diagnostic problem can be defined as follows: Find a mapping from the set of all subsets of observations of a system, to the set of all subsets of possible malfunctions, such that the malfunctions in the subset best explain the observations.

A mathematician's interest in the problem would be satisfied once it can be shown that under certain assumptions this task is *computable*, i.e., an algorithm exists to perform this mapping. He might even further wish to derive the computational complexity of this task for various assumptions about the domain and the range of this mapping. This directly leads to AI algorithms of a set-covering variety for diagnosis (Reggia, et al, 1985).

A logician would consider the solution epistemically complete if he can provide a formalism to list the relevant medical facts and formulate the decision problem as deducing a correct conclusion. Some diagnostic formalisms, such as the ones based on truth maintenance systems, view the diagnostic problem as one more version of truth-maintenance activity (Reiter, 1987).

Now, each of these methods is computationally quite complex, and without extensive addition of knowledge as "heuristics", the problem cannot be solved in anything resembling real time. It is clear, however, that the abstract problem is one that faces intelligent agents on a regular basis: *how to map from states of the world to their explanations?* From the tribesman on a hunt who needs to construct an explanation of observations on the jungle ground to a scientist constructing theories, this basic problem recurs in many forms. Of course, not all versions of the problem are in fact solved by humans, but many versions of the problem, such as medical diagnosis, are solved quite routinely. Presumably something about the agent as an intelligent information processor directly plays a role in this solution process.

Because of our concern in this paper with the structure of intelligence, instead of looking for solutions to this problem in particular domains (such as simple devices, where perhaps tractable algorithms -- e.g., direct-mapping tables that go from symptoms to diagnoses -- might exist and be programmed), let us ask the following question: *What is an intelligence that it can perform this task?* That is, we are interested in the relation between mental structures and the performance of the diagnostic task. The distinction that we are seeking can be made clearer by considering multiplication. Multiplication viewed as a computational task has been sufficiently studied that very fast and efficient algorithms are available, and are routinely used by today's computers. On the other hand if we were to ask, "How does a person (e.g., an arithmetic prodigy) actually perform multiplication in the head?", the answer will be different from the multiplication algorithms just mentioned. The answer would need to be given in terms of how the particular problem is solved by using more generic mental structures. Now, of course, the answer would differ depending upon one's theory of what those mental structures are.

I have already indicated what kinds of answers to this question would be fostered by unitary architectures: In rule-based architectures, the problem solver will simply need to have sufficient

number of rules about malfunctions and observations, frame-theorists would propose that diagnostic knowledge is represented as frames representing domain concepts such as malfunctions, etc. The inference methods that are applicable to each of the above are fixed at the level of the functional architecture: some form of forward or backward chaining for rule systems, and some form of inheritance mechanisms and embedded procedures for frame systems. I have argued elsewhere how this level of abstraction for control is at too low a level to perspicuously encode the inference processes that apply at the level of the task, i.e., diagnosis. (Since all these architectures are computation-universal, they can be made to encode whatever diagnostic algorithm the designer has in mind, but the ideas underlying the algorithms are "lost" as code at these levels, rather than explicitly supported by the architectural constructs.) This level of representation suppresses what is distinctive about diagnosis as a set of domain-independent information processing strategies. See our earlier arguments regarding syntactic solutions at the architectural level.

In our work on knowledge-based reasoning, we have identified several generic strategies, each with a well-defined information processing function. Each of them uses knowledge in certain forms, organized in certain ways, and employs inference strategies that are appropriate to the task. We have described, in a series of papers, how these strategies can be used in different combinations to put together diagnostic or design systems. In the rest of this section, I want to describe briefly how three strategies of the above-described type can come together to solve a number of real world versions of the diagnostic task.

In many domains knowledge is available in the form of malfunction hierarchies (e.g., disease hierarchies in medicine) and for each malfunction hypothesis in the hierarchy, a mapping from observations to the degree of plausibility of hypothesis can be done using a strategy of *concept matching*. In concept matching, a concept is matched to data by a hierarchy of abstractions, each of which produces a degree of local match. In such domains, the diagnostic problem can be decomposed into three subproblems (Chandrasekaran, 1986; Josephson, et al, 1987):

1. **Hierarchical classification:** A classification process on the diagnostic hierarchy is invoked. At the end of the classification process a set of tip nodes of the diagnostic hierarchy are "established" to some degree of plausibility, each explaining a set of observations. In the medical domain, these tip nodes will correspond to specific diseases or in the case of mechanical systems, they may be malfunctions of specific components.
2. **Concept-matching:** Each of the hypotheses in the classification process is evaluated by appealing to the appropriate concept-matching structures which map from relevant data to symbolic confidence value for that hypothesis.
3. **Abductive Assembly:** The classification process (in conjunction with the concept matchers) terminates in a small number of highly plausible hypotheses, each explaining a subset of observations. An *abductive assembly* strategy, which uses knowledge about interaction among malfunctions, can be used to assemble a subset of them into the a composite hypothesis that best explains all the data.

Under the right conditions of knowledge availability, each of the above strategies is computationally tractable. In hierarchical classification, entire subtrees can be pruned if a node is rejected. The mapping from data to concept matching can be done by hierarchical abstractions giving concept matching a similar computational advantage. Abductive assembly can be computationally expensive, but if some other process can prune the space and generate only a small number of hypotheses to begin with, then its computational demand can be kept under control. This is precisely what hierarchical classification does in the above scheme.

The original intractable problem has been converted, by a series of information processing

strategies and by appropriate types of knowledge and control, into a tractable one, *for those versions where knowledge of the required form is available.*

Classification as a strategy is ubiquitous in human reasoning because of the computational advantages of indexing action knowledge over equivalence classes of states, instead of the states themselves. How classification hierarchies are created -- from examples, from other types of knowledge structures, etc. -- requires an additional set of answers. I have discussed elsewhere (Sembugamoorthy and Chandrasekaran, 1986) how knowledge of the relationships between structure and the functions of components, i.e., how the systems work, can often be used to derive such malfunction hierarchies. These processes in turn are generic, requiring knowledge in specific forms and using appropriate but characteristic inference strategies.

Let me make something quite clear at this point. The claim is not that diagnosis is *logically* a classification problem, or even that all human diagnostic reasoning uses classification as one of the strategies. What I have attempted to show is that many version of the diagnostic problem can be, and often are, solved by having knowledge in forms that this and other generic strategies can use. If that knowledge is not available, either other strategies that can generate knowledge in that form are invoked, or other strategies that can help solve the diagnostic problem without classification hierarchies are attempted. In particular, strategies such as *reasoning by analogy* to an earlier case, or merely retrieval of similar cases and explaining the differences by adding, deleting or modifying diagnostic hypotheses are tried. In fact, as mentioned earlier, the whole collection of retrieval strategies (Schank, 1982) are themselves information processing strategies of the functional kind that I have been talking about.

4.7. Functional Theories: Heuristic Becomes Epistemic

What I have so far called functional theories within AI -- GPS, frames as stereotypes, conceptual dependency theory, scripts, and generic tasks in problem solving -- all have this in common: They all typically emphasize some organizational aspect and facilitate some particular kind of inference or construction in a computationally efficient way. In other words, computational feasibility -- the so-called heuristic component -- is built into this kind of theory making. Organization serves directly in securing computational feasibility. A direct epistemic analysis of the underlying problem would typically miss these constructs. Once you discover them you can go back and start doing epistemic analysis on them, but basically the way of discovering them is not by simply taking an epistemic stance toward them (here I use "epistemic" in the McCarthy-Hayes sense of the term).

Another important thing about this with respect to knowledge is that each of these approaches provides primitive terms for encoding the relevant knowledge. GPS proposes that some of the knowledge ought to be encoded in the form of goals and subgoals. The conceptual dependency primitives are provided from the CD theory. Our work on generic tasks has resulted in a family of languages, each of which provides primitives to capture the knowledge needed for that one of the generic strategies. In my view, these primitives constitute part of the vocabulary of the language of thought.

The search for such strategies as the basic task of research in AI in fact defines a new view of epistemics: It is the abstract characterization of such strategies and the corresponding types of knowledge. Such an abstract description of these processes would be replete with terms that carry an information processing strategy connotation, such as *default, goals, subgoals, expectations, plans, and classification.*

5. A proposal on the Nature of Intelligent Information Processing

I have given an overview of the three kinds of theories that have been advanced about the nature of intelligence¹¹:

- architectural theories
- logical abstraction theories, and
- functional theories

and indicated a clear preference for functional theories. I would now like to generalize this preference into a proposal about the nature of intelligence.

The Proposal

Intelligence is a coherent repertoire of generic information processing strategies, each of which solves a type of problem in a computationally efficient way, using knowledge of certain types, organized in a specific way, and using specific and locally appropriate control strategies.

What is common, as intelligent agents, between Einstein, the man-on-the street, the tribesman on a hunt, and, probably, intelligent Alpha-Centaurians (if such things exist) is that they all face very similar computational problems, and the *kinds* of solutions that they adopt for these problems have an essential similarity. They all use plans, indexed by goals, as efficient means of synthesizing actions, they all use some version of scripts and conceptual dependency primitives to organize their inferences, they all use classification strategies to match actions to world states, etc., etc. Of course the strategies that we may discover by studying *human information processing* may not be -- and in all likelihood is not -- coextensive with the general class of such strategies. That would be too anthropomorphic a view of intelligence.

The task of AI as the science of intelligence is to identify these strategies concretely, and understand how they integrate into coherent wholes.

In a sense this approach can be called *abstract psychology* because it doesn't discuss a particular human being or even class of human beings. What it says is that the description of cognitive strategies provides a language in which to describe intelligent agents. And also, I think, it is consistent with the view of intelligence as a biological, evolvable collection of coherent 'kluges' that work together. So intelligence is not really defined as one thing -- architecture or function -- it is really a collection of strategies. The fact that the strategies all contribute to computational feasibility distinguishes them as a characterizable class of information processes.

Some qualifying remarks about the scope of my discussion are perhaps necessary. Almost all my discussion has emphasized cognitive phenomena in contradistinction to perceptual phenomena. Obviously the role of knowledge and control in perception is a much different issue than in cognition. In general, I have not included in my discussion what Fodor (1983) calls *input modules* (as opposed to central processes): his modules include some aspects of parsing in language, e.g. The spirit of what I say in this paper can be extended to these other phenomena, I believe. But that is a task for another day.

¹¹Specifically, "about the nature of 'macro' phenomena in intelligence," since I have acknowledged that connectionist-like theories may have something useful to say about the microstructure.

6. Concluding Remarks

In this final section, I would like to make some remarks about the relationship of functional theories to architectural and abstract characterization theories.

Some of the intuitions behind the architectural theories and abstract characterization theories are in fact valid and useful, but theory-making of these kinds can be enhanced by asking questions of the functional kind first. In particular, considering architectural theories first, it is probably true that there *does* exist an architecture at which mental phenomena can be particularly appropriately implemented. Certainly in the case of human intelligence there *is* some level to which the information processing architecture question can be reduced: if needed, to the neuronal information processing level; possibly to what connectionists call the subsymbolic level; preferably to the level of something like a rule architecture. Certainly I don't intend to argue against the existence of that level of the architecture and its properties. However, the content phenomena of intelligence as a computation are not expressed at that level, and require an analysis at the functional level as I have indicated.

There is another aspect to the architectural issue. To the extent that each of the strategies uses knowledge primitives, and comes with its own inference methods, a local architecture can be associated with it. For example, we have developed a family of high level architectures (or, it comes to the same thing, languages) for the generic information processing strategies that we have identified: a language called CSRL (Bylander and Mittal, 1986) supports hierarchical classification and structured concept matching, PEIRCE (Punch, et al. 1986) supports abductive assembly, and so on. The information processors built using these architectures can exchange the results of their computation with each other.

The functional orientation that I have advocated makes possible a new approach to mental architectures which are nonunitary, i.e., intelligence is conceptualized as a community of "specialists", each of which is an instance of a functional type of knowledge/inference system, communicating with other such entities. In fact our own group's work on problem solving has been implemented using precisely this notion of a nonunitary architecture. It needs to be reiterated that this notion does not argue against the whole set of them being implemented on a lower level unitary architecture, such as a rule architecture. It merely talks about the importance at the conceptual level of not being driven by the unitary architecture notion for all the reasons that we elaborated in Section 4.1 on this subject.

Similarly, the functional theories suggest a new approach to epistemics. They do not argue against the importance of characterizing intelligence independent of incidental implementation considerations (neurons vs transistors, e.g.), or of agent-specific heuristic knowledge (such as the knowledge that a particular agent might have, e.g., "When considering malfunctions in an electronic circuit, always check the power source first.") It is just that this approach proposes an alternative basis on which to make the abstract characterization. I propose that we ask, "What kinds of processes do intelligent agents perform?", rather than, "What kinds of things do they know?" as the starting point. The claim is that what they *need* to know in order to do the tasks in fact provides a new way of doing the epistemic analysis of an agent in the abstract. At the very least, functional theories provide the content information about intelligence as computation that needs to be specified abstractly.

Intelligence as we know it is (so far) a biological phenomenon, rather than a logical or mathematical phenomenon. A comparison is often made between intelligence and flight, and people who would build flying machines by basing them on birds usually come off looking not so good in this comparison. The problem with that analogy is that flying is one (rather well-defined) function,

while intelligence is not characterized by one function. A better analogy would be with understanding life: all we know about life is that it is what biological phenomena pertain to. Any particular aspect of life, e.g. self-reproduction, can be studied mathematically, but there has been precious little so far to show for such studies from the viewpoint of understanding biology. Intelligence is not only analogical to biology, but is, as a phenomenon in nature, so far exhibited in biological organism of certain complexity. The actual content of information processing phenomena of intelligence is bound to be rather complex. What is biological about the proposal in this paper is that intelligence is explained as part evolutionary, part cultural, part life-time interaction and integration of a number of elementary strategies into more and more complex strategies, but all of them are united by this basic concern with computational feasibility for generation of plausible solutions, rather than with deductive correctness. To see biological intelligence as a mere approximate attempt to achieve logical correctness is to miss the point of intelligence completely. Of course there are processes in intelligence that over a long period of time and collectively over humankind help to produce increasingly higher fidelity internal representations, but that is just one part of being intelligent, and in any case such processes are themselves subject to computational feasibility constraints. Once highly plausible candidates for hypotheses about the world are put together by such processes (such as abductive assembly that might be used in producing an explanation in science), then explicit justification processes may be used to verify them.

Acknowledgements: Support by Defense Research Projects Agency through RADC Contract F30602-85-C-0010 and by Air Force Office of Scientific Research, grant 87-0090 during the preparation of this paper is gratefully acknowledged. I would like to thank Dean Allemang, Larry Birnbaum, Tom Bylander, Ashok Goel, Vinod Goel, John and Susan Josephson, Allen Newell, Derek Partridge, Bill Punch, N. Sridharan, and Jon Sticklen for commenting on early drafts. I know that few of them agree with all the things that I say here, so the usual qualification that the author takes responsibility for what is said is particularly relevant.

References

- Bruner, J. S. (1957). On perceptual readiness. *Psychological Review*, 64, 123-152.
- Bylander, T. C., and Mittal, S. (1986). CSRL: a language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7, 3, 66-77.
- Chandrasekaran, B. (1983). Towards a taxonomy of problem-solving types. *AI Magazine*, 4, 1, 9-17.
- Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, Fall, 23-30.
- Chandrasekaran, B. (1986a). From numbers to symbols to knowledge structures: pattern recognition and artificial intelligence perspectives on the classification task, *Pattern Recognition in Practice-II*, E.S. Gelsema and L.N. Kanal (eds), Amsterdam: North-Holland Publishing Company, 547-559.
- Chandrasekaran, B. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. To appear in *Proceedings of the International Joint Conference on Artificial Intelligence*, Milan, Italy.
- Clancey, W. J. and Letsinger, R. (1984). NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. In *Readings in Medical Artificial Intelligence*. W.J. Clancey, E.H. Shortliffe (eds). Reading, MA: Addison-Wesley. 361-381.
- Dennett, D. (1986). The logical geography of computational approaches: a view from the east pole. In Brand, M. & Harnish, R. M. (eds) *The Representation of Knowledge and Belief*, Tucson, AZ, The University of Arizona Press.

- Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*. 12:231-272.
- Dreyfus, H. L. (1979). *What Computers Can't Do*, 2nd Edition. New York: Harper & Row.
- Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York: Wiley-Interscience.
- Fodor, J. A. (1983). *The Modularity of Mind*. Cambridge: The MIT Press, A Bradford Book.
- Fodor, J. A. & Pylyshyn, Z.W. (1987). Connectionism and cognitive architecture: a critical analysis. (draft, to appear).
- Gold, E. (1967). Language identification in the limit. *Information and Control* 16:447-474.
- Gomes F. and Chandrasekaran, B. (1981). Knowledge organization and distribution for medical diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*. SMC-11(1):34-42.
- Hebb, D. O. (1949). *The Organization of Behavior*. New York: Wiley.
- Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C. (1987). A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*. To appear in the Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning.
- McCarthy, J., and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, B. Meltzer and D. Michie (eds). Edinburgh: Edinburgh Univ. Press.
- McClelland, J. L., Rumelhart, D. E., and Hinton, G. E. (1986). The appeal of parallel distributed processing. In Rumelhart, McClelland and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 1*, Cambridge: M.I.T. Press, A Bradford Book.
- McCulloch, W., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. 5: 115-137.
- Miller, G. A., Galanter, E. and Pribram, K. H. (1960). *Plans and the Structure of Behavior*. New York: Holt, Rinehart & Winston.
- Minsky, M. L. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision*, ed. P. H. Winston. New York: McGraw-Hill.
- Minsky, M. L. and Papert, S. (1969) *Perceptrons*, Cambridge, MA: MIT Press.
- Newell, A. (1973). Production systems: models of control structures. In *Visual Information Processing*, ed. W. Chase. New York: Academic Press.
- Newell, A. (1980). Physical symbol systems. *Cognitive Science*, 4, 135-183.
- Newell, A., and Simon, H. A. (1972). *Human Problem Solving*. Englewood Cliffs, N.J.: Prentice-Hall.
- Punch, W., Tanner, M., and Josephson, J. (1986). Design considerations for Peirce, a high level language for hypothesis assembly. In *Proc. Expert Systems In Government Symposium*, Silver Spring, MD: IEEE Computer Society Press, 279-281.
- Pylyshyn, Zenon W. (1984). *Computation and Cognition: Towards a Foundation for Cognitive Science*, The MIT Press, Cambridge, MA, 1984.

- Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*. 12:231-272.
- Dreyfus, H. L. (1979). *What Computers Can't Do*, 2nd Edition. New York: Harper & Row.
- Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York: Wiley-Interscience.
- Fodor, J. A. (1983). *The Modularity of Mind*. Cambridge: The MIT Press, A Bradford Book.
- Fodor, J. A. & Pylyshyn, Z.W. (1987). Connectionism and cognitive architecture: a critical analysis. (draft, to appear).
- Gold, E. (1967). Language identification in the limit. *Information and Control* 16:447-474.
- Gomez F. and Chandrasekaran, B. (1981). Knowledge organization and distribution for medical diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*. SMC-11(1):34-42.
- Hebb, D. O. (1949). *The Organization of Behavior*. New York: Wiley.
- Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C. (1987). A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*. To appear in the Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning.
- McCarthy, J., and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, B. Meltzer and D. Michie (eds). Edinburgh: Edinburgh Univ. Press.
- McClelland, J. L., Rumelhart, D. E., and Hinton, G. E. (1986). The appeal of parallel distributed processing. In Rumelhart, McClelland and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 1*, Cambridge: M.I.T. Press, A Bradford Book.
- McCulloch, W., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. 5: 115-137.
- Miller, G. A., Galanter, E. and Pribram, K. H. (1960). *Plans and the Structure of Behavior*. New York: Holt, Rinehart & Winston.
- Minsky, M. L. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision*, ed. P. H. Winston. New York: McGraw-Hill.
- Minsky, M. L. and Papert, S. (1969) *Perceptrons*, Cambridge, MA: MIT Press.
- Newell, A. (1973). Production systems: models of control structures. In *Visual Information Processing*, ed. W. Chase. New York: Academic Press.
- Newell, A. (1980). Physical symbol systems. *Cognitive Science*, 4, 135-183.
- Newell, A., and Simon, H. A. (1972). *Human Problem Solving*. Englewood Cliffs, N.J.: Prentice-Hall.
- Punch, W., Tanner, M., and Josephson, J. (1986). Design considerations for Peirce, a high level language for hypothesis assembly. In *Proc. Expert Systems In Government Symposium*, Silver Spring, MD: IEEE Computer Society Press, 279-281.
- Pylyshyn, Zenon W. (1984). *Computation and Cognition: Towards a Foundation for Cognitive Science*, The MIT Press, Cambridge, MA, 1984.

Reggia, J., Nau, D., Wang, P., and Peng, Y. (1985). A formal model of diagnostic inference. *Information Sciences*, 37, 227-235.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 32, 57-95.

Rosenblatt, F. (1962). *Principles of Neurodynamics*, Cornell Aeronautical Laboratory Report 1196-G-8. Washington, D.C.: Spartan.

Rumelhart, D.E. & McClelland, J.L. (1986). PDP Models and general issues in cognitive science. In Rumelhart, McClelland and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 1*, Cambridge: M.I.T. Press, A Bradford Book.

Rumelhart, D.E. & McClelland, J.L. (1986b). On learning the past tenses of English verbs. In McClelland, Rumelhart and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 2*, Cambridge, M.I.T. Press. A Bradford Book.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., and Hinton, G. E. (1986). Schemata and sequential thought processes in PDP models. In McClelland, Rumelhart and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 2*, Cambridge, M.I.T. Press, A Bradford Book.

Schank, Roger C. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. New York: Cambridge University Press.

Schank, R. C., and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Hillsdale, N.J.: Erlbaum.

Sembugamorthy, V. and Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem solving systems. In *Experience, Memory and Reasoning*. J. Kolodner and C. Reisbeck (eds). Lawrence Erlbaum Associates, 47-73.

Smolensky, P. (1988). On the proper treatment of connectionism. *The Behavioral and Brain Sciences*, 11, (in press).

Solomonoff, R. J. (1957). An inductive inference machine, *IRE National Convention Record*, pt. 2, 56-62.

Wiener, Norbert. (1948). *Cybernetics, or Control and Communication in the Animal and the Machine*. New York: Wiley.

Appendix K

A Layered Abduction Model of Perception

A Layered Abduction Model of Perception: Integrating Bottom-up and Top-down Processing in a Multi-Sense Agent

John R. Josephson
Laboratory for Artificial Intelligence Research
Ohio State University
Columbus, Ohio 43210

1 February 1989

Abstract

This paper introduces a layered-abduction model of perception which unifies bottom-up and top-down processing in a single logical and information-processing framework. The process of interpreting the input from each sense is broken down into discrete layers of interpretation, where at each layer a "best explanation" hypothesis is formed of the data presented by the layer or layers below, with the help of information available laterally and from above. The formation of this hypothesis is treated as a problem of abductive inference, similar to diagnosis and theory formation. Thus this model brings a knowledge-based problem-solving approach to the analysis of perception, treating perception as a kind of "compiled" cognition.

The bottom-up passing of information from layer to layer defines channels of information flow, which separate and converge in a specific way for any specific sense modality. Multi-modal perception occurs where channels converge from more than one sense.

This model has not yet been implemented, though it is based on systems which have been successful in medical and mechanical diagnosis and medical test interpretation.

Introduction

Computational models of information processing for both vision and spoken language recognition have commonly supposed an orderly progression of layers, beginning near the retina or auditory periphery, where hypotheses are formed about "low-level" features, e.g., edges (in vision) or bursts (in speech perception), and proceeding by stages to higher-level hypotheses. These higher-level hypotheses typically depend largely on hypotheses formed at lower levels, but are also subject to influence from above.

Models intended to be comprehensive often suppose 3 or more major layers, often with sublayers, and sometimes with parallel channels which separate and combine to support higher-layer hypotheses (e.g., shading discontinuities and color contrasts separately supporting hypotheses about object boundary) [31, 28, 29]. Audition, Phonetics, grammar, and semantics have been proposed as layers of interpretation for speech comprehension. Recent work on primate vision appears to show the existence of separate channels for information about shading, texture, and color, not all supplying information to the same layers of interpretation [29].

In both vision and speech understanding most of the processing of information is presumably bottom-up, from information produced by the sensory organ, through intermediate representations, to the abstract cognitive categories that are required for reasoning. Yet top-down processing is significant, as higher-level information is brought to bear to help with identification and disambiguation. Both vision and speech recognition can thus be thought of as "layered interpretation" tasks whereby the output from one layer becomes data to be "interpreted" at the next. Layered interpretation models for non-perceptual interpretive process make

sense too, for example medical diagnosis can be thought of as an inference which proceeds from symptoms, to pathophysiological states, to diseases, to etiologies. It is reasonable to expect that perceptual processes have been optimized over evolutionary time, and that the specific layers and hypotheses, especially at lower levels, have been compiled into special-purpose mechanisms. Perceptual learning provides another source of compilation and optimization. Nevertheless, these layered interpretation models all seem to share certain functional similarities.

In particular, it appears that at each layer of interpretation the information processing task is the same: that of forming a coherent, composite (multi-part) "best explanation" of the data from the previous layer. That is, the task is one of performing an *inference to the best explanation*, in other words, an *abductive inference*.

Moreover, it appears that similar types of hypotheses-hypothesis interactions appear in vision, speech understanding, and diagnosis. Here are three important ones:

1. Two hypotheses might partially overlap in what they can account for, but otherwise be compatible (e.g. an edge might be a boundary for two different objects, the /s/ sound acoustically in the middle of "six stones" belongs to both words, the high white blood count is a result of two different infections),
2. Hypotheses might be pair-wise incompatible (e.g. patch X is either part of the figure or part of the background),
3. Hypotheses might be supportive in an associative way, the presence of one giving some evidence for the presence of the other. (Associative support presumably represents the net impact of several distinct types of evidential relationships.)

The functional similarities suggests the possibility of a generic mechanism, and just such a generic mechanism is proposed here. It is hypothesized that that the processing that occurs in vision, hearing, understanding spoken language, and in interpreting information from other senses (natural and robotic) can all be usefully thought of as variations, incomplete realizations, or compilations (domain-specific optimizations) of this one basic computational mechanism, which we may call *the layered abduction model of perception*.

There is a long tradition of belief that perception involves some form of inference [27] [17] [2]. Several researchers have in fact proposed that perception, or at least language understanding, involves some form of abduction or best-explanation inference [10, p.557] [9] [11] [37] [21, pp.87-94] [14, pp.88,104]. Abduction is often thought of as being logically similar to theory formation in science [17] [46] [14, p.104] and to diagnostic reasoning.

Abduction

The logician and philosopher Charles Sanders Peirce introduced the term "abduction" to refer to a kind of plausible inference, which he took to be logically distinct from both induction and deduction [36]. An abduction passes from a body of data, to a hypothesis that *explains* or *accounts for* that data. Thus abduction is a kind of theory-forming or interpretive inference. In fact Peirce says in one place, "Abductive inference shades into perceptual judgment without any sharp line of demarcation between them." [37, p.304].

In their popular AI textbook Charniak and McDermott characterize abduction variously as *modus ponens* turned backwards, inferring the causal reasons behind something, generation of explanations for what we see around us, and inference to the best explanation [10]. They write that medical diagnosis, story understanding, vision, and understanding natural language are all abductive processes, and they speculate as to whether there might be possible a "unified theory" of abduction" which will link all of these processes together [10, p.557].

Other AI practitioners have given similar characterization of abduction [26] [38] [39] [40], some have proposed or built systems using similar ideas without actually using the term "abduction" in describing their work [3] [33] [12] [35] [41] [34]. Some attempts have been made to cast the natural language understanding problem explicitly as abduction [11] [9]. Philosophers have written of "inference to the best explanation" [19]

[13] [21] and "the explanatory inference" [30]. Related philosophical traditions are the "hypothetico-deductive" model of the scientific method, and accounts of "the logic of discovery" [18]. Recently Paul Thagard has given abduction an important role in his analysis of the logic of scientific theory formation [46].

We may characterize Abduction as a form of inference that follows a pattern like this:

D is a collection of data (facts, observations, givens),
H explains D (would, if true, explain D),
No other hypothesis explains D as well as H does.

Therefore, H is probably true.

The confidence in the conclusion should (and typically does) depend on these factors:

- how decisively H surpasses the alternatives,
- how good H is by itself, independently of considering the alternatives (e.g. we will be cautious about accepting a hypothesis, even if it is clearly the best one we have, if it is not sufficiently plausible in itself),
- how thorough the search was for alternative explanations, and
- pragmatic considerations, including
 - the costs of being wrong and the benefits of being right,
 - how strong the need is to come to a conclusion at all, especially considering the possibility of seeking further evidence before deciding.

I hope my reader recognizes this form of inference as being common in ordinary life, and a part of the "scientific method". What I am proposing here is that it also occurs on many levels in perception.

In general, as Marr pointed out, it is important to distinguish the *goal* of a computation; from the *logic of the strategy* by which that goal can be achieved, from the specific *representations and algorithms* used to describe a specific strategy, and from *implementations* of those representations and algorithms [31, p.25]. Describing a layer of interpretive inference as "abduction" describes the goals of the inference, and suggests strategies to achieve them, as I hope will become clear in what follows. The discussion here will not directly address representation, algorithm, or implementation.

The Layered Abduction Model

Each layer of interpretation, or more precisely, each locus of hypothesis formation (leaving open the possibility of more than one per layer) I call an *agora* after the meeting place where the ancient Greeks would gather for dialog and debate. The picture is that an agora is a place where hypotheses of a certain type gather and contend and where under good conditions a consensus hypothesis emerges. In typical cases the emerging interpretive hypothesis will be a composite hypothesis, coherent in itself, and with different sub-hypotheses accounting for different portions of the data. For example in vision the edge agora can be thought of as the location where a set of edge hypotheses are formed and accepted, each specific edge hypothesis accounting for certain specific data from lower-level agoras.

Our model calls for the information processing at each agora to be decomposed into three functionally distinct types of activity, which we can call *evocation of hypotheses*, *instantiation of hypotheses*, and *composition of hypotheses*.

Evocation can occur bottom-up, a hypothesis being stimulated for consideration by the data presented at the layer below. In diagnosis we would say that the presence of a certain finding *suggests* that certain

hypotheses are appropriate to consider. More than one hypothesis may be suggested by a given datum. Evocation can also occur top-down, either as the result of priming (an expectation from the level above), or as a consequence of data-seeking activity from above, which can arise from the need for evaluation. Evocations can in general be performed in parallel, and need not be synchronized.

Instantiation occurs when each stimulated hypothesis is independently scored for confidence (*evaluation*), and a determination is made of what part or aspect of the data the hypothesis can account for (*determination of explanatory scope*). This process is in general top-down, and in order to instantiate itself a hypothesis may seek data which was not part of its original stimulus¹. The data which are accounted for may or may not be identical to the data upon which the hypothesis was scored, or the data which did the evoking.

In the course of instantiation the hypothesis set may be expanded by including subtypes and supertypes of high-confidence hypotheses². Instantiation is typically based on matching against prestored patterns of features, but instantiating "by synthesis" is also possible whereby the features to match are generated at run time. The result of a wave of hypothesis instantiation is a set of hypotheses, each with some measure of confidence, and each offering to account for some portion of the data. Usually many of the evoked hypotheses can be ruled out, and will not form part of the result. Since in a wave of instantiation hypotheses are considered independently of each other, this too can go on in parallel.

Composition occurs when the instantiated hypotheses interact with each other and (under good conditions) a coherent best interpretation emerges. Note that, as this stage begins, each hypothesis has both a confidence value, and a body of data that it can account for. In the end some hypotheses will have been incorporated into the composite hypotheses, some will have been excluded, and perhaps some will be in limbo as a result of some remaining ambiguity of interpretation.

Strategy for Composition of Hypotheses

For hypothesis composition an overall abduction problem has been set up: to account for all of the (reliable and important) data presented by the agora(s) immediately below. A series of small abduction problems is also set up: to account for each particular datum. A basic strategy is to try to solve the overall abduction problem by solving a sufficient number of smaller and easier abduction problems. We begin by solving the easiest small abduction problems, the ones in which we can have the most confidence. If a certain hypothesis is *the only plausible explanation* for some finding (it accounts for the finding and its local-match confidence value is not too low), then it is entitled to high confidence, and entitled to be accepted into the overall composite hypothesis that represents the solution to the overall abductive problem.

Let us call a hypothesis "BELIEVED" when it has been accepted as the correct interpretation for the data it offers to account for. Data accounted for by BELIEVED hypotheses are "ACCOUNTED-FOR" and are considered to be successfully interpreted. Let us call a hypothesis "ESSENTIAL" if it is the only plausible explanation for some reliable datum (which is typically a hypothesis at the next lowest level that is BELIEVED). Thus an ESSENTIAL hypothesis scores positively and accounts for data items for which there are no other good interpretations. ESSENTIAL hypotheses are BELIEVED. Information about the explanatory relationships is thus used to increase the confidence in certain hypotheses.

If not all of the data are yet accounted for, the next step is to propagate the consequences of the initial set of BELIEVED hypotheses. These consequences arise as the result of causal and statistical relationships between hypotheses typically stored as compiled knowledge in advance of processing. There are several kinds of these relationships—I describe them here just briefly. Hypotheses at the same level (in the same agora) can have relationships of compatibility, entailment, or incompatibility, which can be a matter of degree. Propagating the consequences of BELIEVED hypotheses by taking account of these relations requires the appropriate adjustment of scores for related viable hypotheses outside of the BELIEVED set, or other appropriate actions. For example a hypotheses incompatible with a BELIEVED hypotheses can be rejected categorically, and removed from further consideration. Another kind of relationship is where a hypotheses "EXPECTS" the

¹ Under certain data-driven circumstances it is good enough just to score on the basis of voting by the stimulating data from below, and then no top-down processing need occur, at least for scoring.

² In general the space of potential hypotheses can be assumed to be hierarchically organized by level of specificity.

presence of data items at the next lower level (this too can come in degrees). Propagating such an expectation requires evoking the hypothesis corresponding to the expectation (priming) if it has not already been evoked. If it has, it can be given an extra measure of confidence. If a strong expectation is *contradicted* by the data, an anomaly has occurred, and special handling is appropriate.³

A hypothesis is a "CLEAR-BEST" if it is the distinctly best explanation (by confidence level) of some data item. CLEAR-BEST hypotheses are BELIEVED too. Note that an ESSENTIAL or CLEAR-BEST hypothesis is the uniquely best explanation for some data items—it is a local abductive conclusion. If not all of the data has been accounted for, the consequences of CLEAR-BESTs are propagated similarly to the ESSENTIAL hypotheses. Note that this propagation can result in more hypotheses becoming CLEAR-BESTs, e.g., if high-scoring explanatory competitors are removed from consideration, or if propagating consequences readjusts hypothesis scores so that a clear winner emerges.

If the ESSENTIAL hypotheses together with the CLEAR-BESTs do not account for everything, we have done all we can do on the current evidence without resorting to guessing. Generally our best strategy under these circumstances would be to go back for more data. In fact we are in a position to guide the data gathering by focusing on the problem of discriminating between alternative good explanations for significant data items. This is a form of top-down processing we may call "focused disambiguation". Sometimes, however, we have all of the relevant data we are going to get, for example we may be unable to ask the speaker to repeat. Under these circumstances we still have the means available to do some clever guessing. We can begin to include hypotheses which are best explanations for certain findings, but which are not far enough ahead of the alternatives, or not of high enough local-match confidence, to enable them to be accepted confidently. These WEAKLY-BESTs constitute the best guesses we can make under the circumstances. Actually some of them can be accepted with a fairly high degree of confidence. A finding can be made to vote for the hypotheses which best explain it (with voting strength in proportion to the measure by which the hypothesis beats its nearest competitor). The idea is that two different findings, both pointing to the same hypotheses as the best explanation constitute (apparently) independent sources of evidence for the hypothesis, i.e., constitute converging lines of inference for the hypothesis. Hypotheses with more votes can be accepted more confidently than hypotheses with fewer votes, and perhaps enough can be confidently accepted to complete the explanation.

Now in general relationships (spatial, grammatical, etc.) between the parts of a hypothesis are significant and need to be maintained. Some of these relationships can be seen as the filling of related roles in higher-layer interpretive hypotheses, for example a diagnostic hypothesis of a flow going on between A and B would bind A-related and B-related data together into relationships. But some other relationships (e.g. spatial in low level vision) are presumably compiled into the hardware, so that the appropriate constraints are applied between neighboring hypotheses as an automatic result of the operation of the machinery. Still and all, the net impact on hypothesis composition of these relationships can probably be captured by basic relationships of mutual sympathy and antipathy.

At the end of a wave of composition activity certain hypotheses have been accepted as BELIEVED. These constitute a confident best explanation for a portion of the data. Often there will also remain a set of unexplained data, and a set of viable hypotheses which, at various levels of confidence, offer to explain that data, but for which no clear solution is apparent. Nevertheless the BELIEVED hypotheses may be enough data for the next higher layer to do its business; resolving the remaining ambiguities may be unimportant in the context. Alternatively, remaining ambiguities may get resolved later as a result of further processing at that layer stimulated by downward-flowing expectations.

Downward-Flowing Processing

We may distinguish at least four sources or functions of top-down processing. One is that the data-seeking needs of hypothesis evaluation can provoke computation of the data (top-down evocation and evaluation of a hypothesis) as was discussed above. Another that was mentioned is that expectations based on firmly established hypotheses at one layer can prime certain data items (i.e. evoke consideration of them and bias

³ Throughout the processing various kinds of anomalies can occur. Anomalies are detected and recorded, and typically stimulate special handling; from here on I describe the course of processing only for when everything goes smoothly.

their score upwards). A third way is that hypotheses that are uninterpretable as data at the higher level (no explanation can be found) can be "doubted" and reconsideration of them provoked. Finally data pairs that are jointly uninterpretable, as for example two words, the co-occurrence of which cannot be reconciled syntactically or semantically, can be considered to be incompatible (to some degree of strength) and recomputation of the composite hypothesis can be provoked from above. In these ways higher-level interpretations can exert a strong influence on the formation of hypotheses at lower levels, and layer-layer harmony is a two way street.

Recovering from Mistakes

Mistakes in Initial Hypothesization and Scoring

- Hypothesis suggestions come from above as well as below, thus hypotheses which would be missed on bottom-up processing can still be considered.
- If suggestions are inadequate, e.g. no hypotheses are evoked covering a segment of data, or all suggestions score low, exhaustive search (though hierarchically organized for efficiency) is undertaken to broaden the hypotheses being considered, thus hypotheses that are missed on suggestion-based stimulation can still be considered.
- Hypothesis evaluation is augmented by encouragement and discouragement (resulting from positive associations and incompatibilities) from other hypotheses in the same agora. Thus the local-match confidence score is improved by contextual information.
- Hypotheses evaluation is augmented by encouragement and discouragement based on expectations derived from confident higher-level hypotheses. This constitutes another kind of context-based improvement and check on the the confidence score.
- The acceptance of a hypothesis is based on how well it surpasses explanatory alternatives, thus after recognition-based scoring, a significant additional uncertainty-reducing operation is performed before acceptance.
- Strength of confidence is supported by "the consilience of inductions" whereby converging lines of inference all support the same hypotheses. Thus system performance should be robust.
- Acceptance, when it finally occurs, is still tentative and liable to be overthrown by relationships to the mass of other confident hypotheses.

Mistakes in Choice of Initial Islands of Confidence

- Actually the islands are very strong. They are never based only on a hypothesis having high initial confidence; it is at least required to also be a distinctly best explanation for some datum.
- Inconsistencies lead to detected anomalies, which lead to special strategies that weigh alternative courses of action. Originally accepted hypotheses can collide with others and subsequently called into question.
- Inconsistency collisions can occur laterally, or from above (violation of expectation, or from below (violation of expectation), and can come in degrees of strength. In effect there is broad cross checking of accepted hypotheses.
- An inexplicable datum should be doubted and called into question—it may not really be there. If after re-evaluation the datum remains strong despite the doubt, then the system can detect that it has encountered the limits of its knowledge, and is positioned to learn a new hypothesis category.
- Sometimes two parts of a compound hypothesis are inconsistent in context, where the judgment of higher levels is that they cannot both occur, based upon the inability to form a consistent hypothesis at the next highest level. (It seems that this can account for unstable perceptual objects like the Necker cube.)

Summary of the Control Strategy

We may summarize the control strategy by saying that it employs multi-level and multiple intra-level island-driven processing. Islands of relative certainty are seeded by local abductions and propagate laterally (incompatibilities, positive associations), downwards (expectations), and upwards (firm items of data to be accounted for). Processing occurs concurrently and in a distributed fashion. Higher levels provide *soft* constraints through the impact of expectations on hypothesis evocation and scoring, but do not strictly limit the hypothesis space.

Extension of the Model to Multi-Modal Perception

The basic idea in extending the model to multi-modal perception, i.e. perception that combines the information from more than a single sense, is that combining information from different senses is functionally no different than combining information from different channels within one sense modality. Different channels within the visual system deliver up the data useful at a certain level to form hypotheses about the locations of 3-d objects within the visual space; similarly, different senses deliver up the data useful for forming hypotheses about, say, object identity.

One special processing problem for multi-sense integration is the problem of identifying a "That" delivered up by one sense, with a "That" delivered up by another. Which person is the one that is speaking? Is it the same object being seen in the infrared as that being seen in x-rays? Logically, it should be possible for information derived from one sense to help with resolving distinct objects within the other sense. There is actually some evidence that vision can help hearing to separate distinct streams of tones [32, p.83] and hear the tone stream as two distinct auditory objects.

One useful computational support for cross modal perception is provided by correlated spatial representations, as our visual maps are correlated with our auditory maps of the space surrounding us. Thus, for example, a robot should bring together separate channels of information from its senses of "sight" and "touch" into a unified spatial representation of its immediate surroundings. Moreover this "hot map" of its surroundings should be maintained continually, and updated and revised as new information arrives and is interpreted. This hot map, with its symbols on it, can be viewed as the resulting composite hypothesis formed at "the agora of objects in the immediate surroundings" by a process of abductive interpretation.

Yet some senses are not particularly spatial (e.g. smell). We can envision computational support for cross-modal perception in the form of pattern-based recognition knowledge, where the compiled recognition patterns for an object category rely on features from more than one sense. This is very analogous to medical diagnosis where a disease is recognized from evidence from such disparate sources as lab tests, x-rays, and patient history. Such recognition knowledge can be used to support an "agora of the patient's disease" in much the same manner as the robot mentioned above maintains its map of objects in its surroundings. Somewhat further along we can envision a robot that maintains an "agora of understanding" whereby it monitors some complex device and continually maintains a causal understanding of its. Much much further along we can imagine building a robot scientist who maintains an "agora of theoretical understanding" whereby its best understanding of the world is maintained.

Summary: Perception as Compiled Cognition

The formation of a composite best-explanation hypothesis at any level in perception is treated as a problem of abductive inference, similar to diagnosis and theory formation. Thus this model brings a knowledge-based problem-solving approach to the analysis of perception, treating perception as a kind of "compiled" cognition.

Acknowledgments

The model described here is an outgrowth of research on diagnostic reasoning that has gone on at Ohio State Laboratory for AI Research over the past decade in the context of a developing theory of generic information

processing tasks [4], [6], [5], [7], [8], [42], [43], [44]. Various facets of abductive reasoning have been investigated, especially the problem of assembling composite explanatory hypotheses [24], [25], [26], [1], [20], [16], [15], [45], [23], [22].

This work has been supported by the Defense Advanced Research Projects Agency under RADC contract F30602-85-C-0010; and the NIH Heart, Lung and Blood institute, grant number 1 RO1 HL38776-01. The development of the layered abduction model has benefited especially from discussions with B. Chandrasekaran, Susan Josephson, Dean Allemang, Ashok Goel, Tom Bylander, Michael Tanner, Terry Patten, and Jordan Pollack, as well many discussions at the Image Understanding Workshop sponsored by DARPA/ISTO in April 1988.

References

- [1] D. Allemang, M. C. Tanner, T. Bylander, and J. R. Josephson. On the computational complexity of hypothesis assembly. In *Proc. Tenth International Joint Conference on Artificial Intelligence*, Milan, August 1987.
- [2] Jerome S. Bruner. On perceptual readiness. *Psychological Review*, 64(2):123-152, 1957.
- [3] B. G. Buchanan, Sutherland, G. L., and E. A. Feigenbaum. *Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry*, pages 54-69. Edinburgh University Press, Edinburgh, 1969.
- [4] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, pages 23-30, 1986. Fall 1986.
- [5] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987.
- [6] B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. Technical report, The Ohio State University, 1988.
- [7] B. Chandrasekaran, F. Gomez, S. Mittal, and J. W. Smith. An approach to medical diagnosis based on conceptual structures. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 134-142, 1979.
- [8] B. Chandrasekaran and S. Mittal. Conceptual representation of medical knowledge for diagnosis by computer: Mdx and related systems. In M. Yovits, editor, *Advances in Computers*, volume 22, pages 217-293. Academic Press, 1983.
- [9] E. Charniak. A neat theory of marker passing. In *Proceedings of AAAI-86*, volume 1, pages 584-588. AAAI, Morgan Kaufmann, August 1986.
- [10] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence*. Addison Wesley, 1985.
- [11] Venugopala Rao Dasigi. *Word Sense Disambiguation in Descriptive Text Interpretation: A Dual-Route Parsimonious Covering Model*. PhD thesis, University of Maryland, College Park, 1988.
- [12] Johan de Kleer. Reasoning about multiple faults. *AI Magazine*, 7(3):132-139, August 1986.
- [13] R. Ennis. Enumerative induction and best explanation. *The Journal of Philosophy*, LXV(18):523-529, September 1968.
- [14] Jerry Fodor. *The Modularity of Mind*. Bradford Book, 1983.
- [15] Ashok Goel, J. Ramanujan, and P. Sadayappan. Towards a 'neural' architecture for abductive reasoning. In *Proceedings of the Second International Conference on Neural Networks*, volume 1, pages 681-688, 1988.

- [16] Ashok Goel, P. Sadayappan, and John R. Josephson. Concurrent synthesis of composite explanatory hypotheses. In *Proceedings of the Seventeenth International Conference on Parallel Processing*, pages 156-160, august 1988.
- [17] Richard L. Gregory. Perception as hypotheses. In Richard L. Gregory, editor, *The Oxford Companion to the Mind*, pages 608-611. Oxford University Press, 1987.
- [18] N.R. Hanson. *Patterns of Discovery*. Cambridge University Press, 1958.
- [19] Gilbert Harman. The inference to the best explanation. *Philosophical Review*, LXXIV:88-95, January 1965.
- [20] John Josephson. A framework for situation assessment: Using best-explanation reasoning to infer plans from behavior. In *Proceedings of Expert Systems Workshop*, pages 76-85, April 1987.
- [21] John R. Josephson. *Explanation and Induction*. PhD thesis, The Ohio State University, 1982.
- [22] John R. Josephson. Reducing uncertainty by using explanatory relationships. In *Proceedings of the Space Operations Automation and Robotics Conference-1988 (Soar-88)*, pages 149-151, 1988.
- [23] John R. Josephson. Towards a generic architecture for layered interpretation tasks—implications for plan recognition. Technical report, The Ohio State University, 1988.
- [24] John R. Josephson, B. Chandrasekaran, and Jack Smith, Jr. Assembling the best explanation. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, pages 185-190, Denver, Colorado, December 1984. IEEE Computer Society. A revised version by the same title is available as a technical report.
- [25] John R. Josephson, B. Chandrasekaran, Jack Smith, Jr., and Michael C. Tanner. Abduction by classification and assembly. In *PSA 1986 Volume One*, volume 1, pages 458-470, East Lansing, Michigan, 1986. Philosophy of Science Association.
- [26] John R. Josephson, B. Chandrasekaran, Jack Smith, Jr., and Michael C. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning*, SMC-17(3):445-54, May, June 1987.
- [27] Immanuel Kant. *Critique of Pure Reason*. St. Martins Press, (1968), New York, 1787. tr. Norman Kemp Smith.
- [28] V.R. Lesser, R.D. Fennell, L.D. Erman, and R.D. Reddy. The Hearsay II speech understanding system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-23:11-24, 1975.
- [29] Margaret Livingstone and David Hubel. Segregation of form color, movement, and depth: Anatomy, physiology, and perception. *Science*, 240, May 1988.
- [30] William G. Lycan. Epistemic value. *Synthese*, 64:137-164, 1985.
- [31] David Marr. *Vision*. W. H. Freeman and Company, 1982.
- [32] Dominic William Massaro. *Speech Perception by Ear and Eye: A Paradigm for Psychological Inquiry*. Lawrence Erlbaum Associates, New Jersey, 1987.
- [33] Randolph A. Miller, Harry E. Pople, Jr., and Jack D. Myers. Internist - i, an experimental computer-based diagnostic consultant for general internal medicine. *New England Journal of Medicine*, 307:468-476, 1982.
- [34] R. S. Patil. *Causal representation of patient illness for electrolyte and acid-base diagnosis*. Ph. D Thesis, 1981.
- [35] J. Pearl. Distributed revision of composite beliefs. *Artificial Intelligence*, 33(2):173-215, October 1987.
- [36] Charles S. Peirce. Abduction and induction. In Justus Buchler, editor, *Philosophical Writings of Peirce*, chapter 11, pages 150-156. Dover, 1955.

- [37] Charles S. Peirce. Perceptual judgments. In Justus Buchler, editor, *Philosophical Writings of Peirce*, pages 302-305. Dover, 1955.
- [38] H. Pople. On the mechanization of abductive logic. *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 147-152, 1973.
- [39] James Reggia. Abductive inference. In Kamal N. Karna, editor, *Proceedings of The Expert Systems in Government Symposium*, pages 484-489. IEEE Computer Society Press, 1985.
- [40] James A. Reggia, Barry T. Perricone, Dana S. Nau, and Yun Peng. Answer justification in diagnostic expert systems-part 1: Abductive inference and its justification. *IEEE Transactions on Biomedical Engineering*, BME-32(4):263-267, April 1985.
- [41] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57-95, 1987.
- [42] Jack Smith, Jr., John R. Svirbely, Charles A. Evans, Pat Strohm, John R. Josephson, and Michael C. Tanner. Red: A red-cell antibody identification expert module. *Journal of Medical Systems*, 9(3):121-138, 1985.
- [43] Jack W. Smith, Jr. *RED: A Classificatory and Abductive Expert System*. PhD thesis, Ohio State University, August 1985.
- [44] Jon Sticklen. *MDX-2: An Integrated Medical Diagnostic System*. PhD thesis, The Ohio State University, 1987.
- [45] Mike Tanner and John Josephson. Abductive justification. Technical report, Ohio State University Laboratory for AI Research, 1988.
- [46] Paul Thagard. *Computational Philosophy of Science*. Bradford Books / MIT Press, 1988.

Appendix L

Papers and Technical Reports

What follows is a list of papers and technical reports that were supported wholly or partially by the project. Abstracts are included for some of them. Copies or reprints are available on request with a nominal charge to cover costs. Write to: LAIR Technical Reports, CIS Dept., The Ohio State University, 228 Bolz Hall, 2036 Neil Avenue Mall, Columbus, OH, 43210, U.S.A. Please include the symbolic code for any reports requested. The symbolic code is the item associated with each entry which appears between the [square brackets].

1. B. Chandrasekaran, Generic Tasks in Expert System Design and Their Role in Explanation of Problem Solving, appears in *Proceedings of the Workshop on AI and Distributed Problem Solving*, National Academy of Sciences Office of Naval Research, 1985, May 16-17. Also appeared in in *Proceedings of the Expert Systems Workshop*, DARPA, April 1986, pages 127-135. [85-BC-EXPROB] (This paper has been superseded by [87-BC-EXPCON].)
2. B. Chandrasekaran, J. Josephson and A. Keuneke, Functional Representations as a Basis for Generating Explanations, appeared as an invited paper in *Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics*, October 14-17, 1986, IEEE, Atlanta, Georgia. [86-

BC-FUNEXPL]

ABSTRACT: It is generally agreed that systems used to provide consultation or advice need to explain their knowledge and problem solving in order to be acceptable and useful. We have been developing a framework that unifies problem solving, deep cognitive models, and explanation. We have earlier proposed a functional representation for modeling understanding of how devices work. In this paper we outline how plans can be thought of as abstract devices, and how aspects of a planner's behavior can be explained by using this representation.

3. B. Chandrasekaran, Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, appeared in *IEEE Expert*, Fall 1986, pages 23-30. [86-BC-IEEEX]

ABSTRACT: In the view of our research group at the Laboratory for Artificial Intelligence Research, the field of expert systems is stuck in a level of abstraction that obscures the essential nature of the information processing tasks that current systems perform. The available paradigms often force us to fit the problem to the tools rather than fashion the tools to reflect the structure of the problem. This situation is caused by a failure to distinguish between what we might call the information processing level (or the knowledge level, in Allen Newell's words) and the implementation language level. Most available languages, be they rule-, frame-, or logic-based, are more like assembly languages than high-level programming languages with constructs essential for capturing the essence of the information processing phenomena. We wish to provide a critique of the abstraction level of the currently dominant approaches and propose an alternative level of abstraction. The proposed alternative not only clarifies the issues but also makes possible tools and approaches that help in system design, knowledge acquisition, and explanation.

4. B. Chandrasekaran and J. Josephson, Explanation, Problem Solving, and New Generation Tools: A Progress Report, appears in the *Proceedings of the Expert Systems Workshop*, April 1986, Science Applications International Corporation under contract to DARPA, April, 1986, Pages 101-126. [86-BC-NEWGEN]

ABSTRACT: This is a progress report on our project on "Explanation in Planning and Problem Solving Systems". It is being written approximately at the 15-month mark. Conceptual frameworks for generation of explanation of two kinds have been built: one for explaining how decisions are

made during problem solving, explaining control strategies as well as other aspects of run-time behavior, and the other to give a planner the capacity to represent an understanding of its own plan fragments, and thus to explain to the user how a plan is meant to work. A prototype mission planning system with some explanation capabilities has been built, and a number of high-level knowledge-based system construction tools have been built with features that facilitate knowledge acquisition, system implementation and explanation generation. Two of these tools (DSPL and HYPER) are discussed in this report, one (CSRL) predates this explanation project and has been extensively reported on, and several others are in various stages of design and implementation. Together they will constitute a high-level tool box for the construction of knowledge-based systems. They will be useful for building a variety of planning, diagnostic, abductive, and retrieval systems, and systems which are combinations of these types. These tools have as design features a number of "hooks" for the attachment of explanation synthesis tools.

In the first stage of this project, we have chosen "routine planning" as a task for which to build a prototype. In particular, a planning task for Offensive Counter Air (OCA) missions was chosen for analysis and implementation.

5. B. Chandrasekaran, J. Josephson, A. Keuneke and D. Herman, An Approach to Routine Planning, appeared in *Proceedings of the Knowledge-Based Planning Workshop*, Austin, Texas, December 1987, pages 25.1-25.10. [86-BC-PLANNING] (This paper has been superseded by [86-BC-PLANEXPLAIN].)
6. B. Chandrasekaran, J. Josephson, A. Keuneke and D. Herman, Building Routine Planning Systems and Explaining Their Behavior. This paper supersedes [86-BC-PLANNING] and is to appear in the *International Journal of Man-Machine Studies* in 1989. [86-BC-PLANEXPLAIN]

ABSTRACT: It has become increasingly clear to builders of knowledge-based systems that no single representational formalism or control construct is optimal for encoding the wide variety of types of problem solving that commonly arise and are of practical significance. In this paper we identify a class of problem solving activities which we have labeled routine planning. We consider the constructs necessary to represent the problem solving which appropriately characterizes this class, and describe DSPL, a high-level language designed specifically to encompass the required knowledge structures and control methodology for routine planning. Finally, we

consider what type of structure is appropriate to represent an agent's understanding of how the plan itself works.

7. B. Chandrasekaran and A. Keuneke, Classification Problem Solving: A Tutorial From An AI Perspective, appears as an invited paper on "Pattern Recognition Theory and Applications" in *Proceedings of the NATO Advanced Study Institute*, June 8-20, 1986, Spa, Belgium; appears as chapter 31 of *Pattern Recognition Theory and Applications*, Springer-Verlag Publishers, edited by P. Devijver and J. Kittler, pp. 393-409, 1986. [86-BC-TUTORI]
8. B. Chandrasekaran and M. Tanner, Uncertainty Handling in Expert Systems: Uniform Vs. Task-Specific Formalisms, appears in *Uncertainty in Artificial Intelligence*, pages 35-46, Editors, L. N. Kanal and J. Lemmer, Publisher, North Holland Publishing Company, 1986. [86-BC-UNCERT]
9. Jesse F. Dillard, Kamesh Ramakrishna, B. Chandrasekaran, Knowledge-Based Decision Support System for Military Procurement, appears in *Expert Systems for Business*, Barry G. Sliverman, editor, Addison-Wesley, 1987, pp. 120-139. [86-JD-MILPRO]
10. S. Hashemi, B. Hajek, D. Miller, B. Chandrasekaran and J. Josephson, Expert Systems Application to Plant Diagnosis and Sensor Data Validation, appears in *Proceedings of The Sixth Power Plant Dynamics, Control and Testing Symposium*, Knoxville, Tennessee, April, 1986. [86-SH-SENSOR]
11. D. Herman, J. Josephson and R. Hartung, Use of DSPL for the Design of a Mission Planning Assistant, appears in *Proceedings of the IEEE Expert Systems in Government Symposium*, IEEE Computer Society, October, 1986, Pages 273-278. [86-DH-DSPL]
12. W. Punch and J. Josephson, A Classification Approach to the Diagnosis of Malfunctions in Complex Mechanical Systems, OSU CIS LAIR Technical Report. [86-WP-MALFUN]

ABSTRACT: Classification problem solving provides a powerful methodology for diagnosis of faults in complex mechanical systems. Hierarchical representations of such systems organize the problem solving so that is is modular and computationally efficient. This approach incorporates knowledge of four related but distinct types; matching knowledge, accounts-for knowledge, expectation knowledge and decomposition knowledge. How

these knowledge types are used in diagnosis is discussed, as well as an overall system architecture. Examples from existing systems are provided.

13. W. Punch, M. Tanner and J. Josephson, Design Considerations for Peirce, a High Level Language for Hypothesis Assembly, appears in *Proceedings Expert Systems In Government Symposium*, IEEE Computer Society Press, 1986, October, Pages 279-281. [86-WP-PEIRCE]

ABSTRACT: Diagnosis is essentially a problem of abduction, that of inferring a best explanation for a body of data. This has been recognized in the design of many expert systems (RED, Internist, Dendral). Generating and evaluating explanatory hypotheses, in order to satisfy explanatory goals, is clearly a pervasive form of knowledge based reasoning. In this paper we describe a particular strategy for abduction: the assembly of a complete explanation from a set of simpler hypotheses. We identify the types of knowledge needed for abductive-assembly, and discuss design considerations for a programming tool (Pierce) for building expert systems and subsystems which perform abductive assembly.

14. D. Allemang, M. Tanner, T. Bylander and J. Josephson, On the Computational Complexity of Hypothesis Assembly, appears in the *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August, 1987, Milan, Italy. [87-DA-ABDCOMP]
15. T. Bylander and B. Chandrasekaran, Generic Tasks for Knowledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition, appears in *International Journal of Man-Machine Studies*, 1987, 26, pp. 231-243. This is a revised edition of an earlier paper by the same title which appeared in *Proceedings of the Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, November, 1986. [87-TB-KNOWAC]

ABSTRACT: Our research task has been to identify generic tasks-basic combinations of knowledge structures and inference strategies that are powerful for solving certain kinds of problems. Our strategy is best understood by considering the "interaction problem", that representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and by the inference strategy to be applied to the knowledge. The interaction problem implies that different knowledge-acquisition methodologies will be required for different kinds of reasoning, e.g. a different knowledge-acquisition methodology for each generic task. We illus-

trate this using the generic task of hierarchical classification. Our proposal and the interaction problem call into question many generally held beliefs about expert systems such as the belief that the knowledge base should be separated from the inference engine.

16. T. Bylander, B. Chandrasekaran and J. Josephson, The Generic Task Toolset, appears in *Proceedings of the Second International Conference on Human-Computer Interaction*, Honolulu, Hawaii, August, 1987. [87-TB-TOOLSET] Included as Appendix H to this report.
17. B. Chandrasekaran, What Kind of Information Processing Is Intelligence? A Perspective On AI Paradigms and a Proposal, to appear in *Source Book on the Foundations of AI*, Partridge and Wilks, Editors, Cambridge University Press, 1988. [87-BC-PARADI] Included as Appendix J to this report.
18. B. Chandrasekaran, A. Goel and D. Allemang, Connectionism and Information Processing Abstractions: The Message Still Counts More Than the Medium, appears in *The Journal of Behavioral and Brain Sciences*, 1988, as a commentary on Paul Smolensky's paper entitled "On the Proper Treatment of Connectionism"; appears in *AI Magazine*, Winter 1988, pp. 24-34; [87-BC-CONNECT] (Expanded version of parts of [87-BC-PARADI], this paper is superseded by [88-BC-CONPROAB].)
19. B. Chandrasekaran, J. Josephson and K. Schwan, Artificial Intelligence and Concurrent Processing, OSU CIS LAIR Technical Report. [87-BC-CONPRO]
20. B. Chandrasekaran, M. Tanner and J. Josephson, Explanation: The Role of Control Strategies and Deep Models, appears in *Expert Systems: The User Interface*, Ablex Press, James Hendler, ed., 1987. [87-BC-EXPCON] (This paper subsumes [85-BC-EXPROB] and [87-BC-EXPLAN] and has been superseded by [89-BC-EXPLAN].)
21. B. Chandrasekaran and W. Punch, Hierarchical Classification: Its Usefulness for Diagnosis and Sensor Validation, in *Proceedings of the Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program*, March 9-11, Arlington, Virginia. [87-BC-HIERAR] (This paper subsumes [85-BC-SENCON] has been superseded by [87-BC-VALIDA].)

22. B. Chandrasekaran, Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks, appears as an invited address in the *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 28, 1987. [87-BC-IJCAI]

ABSTRACT: The level of abstraction of much of the work in knowledge-based system (the rule, frame, logic level) is too low to provide a rich enough vocabulary for knowledge and control. I provide an overview of a framework called the Generic Task approach that proposes that knowledge systems should be built out of building blocks, each of which is appropriate for a basic type of problem solving. Each generic task uses forms of knowledge and control strategies that are characteristic to it, and are generally conceptually closer to domain knowledge. This facilitates knowledge acquisition and can produce a more perspicuous explanation of problem solving. The relationship of the constructs at the generic task level to the rule-frame level is analogous to that between high level programming languages and assembly languages. I describe a set of generic tasks that have been found particularly useful in constructing diagnostic, design and planning systems; diagnostic reasoning is used to illustrate the approach. I describe the Generic Task Toolset for constructing knowledge systems, which embodies the Generic Task approach. I conclude with the implications of this approach for the functional architecture of intelligence.

23. B. Chandrasekaran, J. Smith and J. Sticklen, "Deep" Models and Their Relation to Diagnosis, Invited paper, Toyota Foundation Symposium on Artificial Intelligence in Medicine, Tokyo, Japan, August 1986. OSU CIS LAIR Techreport [87-BC-JAPAN] (Superseded by [88-BC-JAPAN].)
24. B. Chandrasekaran, J. Josephson and D. Herman, The Generic Task Toolset: High Level Languages for the Construction of Planning and Problem Solving Systems, appears in *Proceedings of the Workshop on Space Tele-robotics*, NASA and Jet Propulsion Laboratories, Pasadena, California, January 20-22, 1987. [87-BC-TOOLEV] (Superseded by [88-BC-ROUTINE].)
25. B. Chandrasekaran and W. Punch, Data Validation During Diagnosis: A Step Beyond Traditional Sensor Validation, appears in *Proceedings of AAAI-87*, pp. 778-782. [87-BC-VALIDA]

ABSTRACT: As diagnostic systems reach greater acceptance solving real world problems more attention must be paid to the system's ability to deal with conflicting data. That is, human experts show an ability to reach diag-

nostic conclusions using conflicting or absent data while many diagnostic expert systems require correct data for correct conclusions. As an example we show that the domain of mechanical systems traditionally approaches the problem by validating system with various kinds of redundancy in sensor hardware before doing diagnosis. While such techniques are useful, we propose that another level of redundancy provided by expectations derived during diagnosis. We further show how such expectation-based data validation is a natural part of diagnosis as performed by hierarchical classification expert systems.

26. A. Goel, N. Soundararajan and B. Chandrasekaran, Complexity in Classificatory Reasoning, appears in *Proceedings of AAAI-87*, July 13-18, Seattle, Washington, Pages 421-425. [87-AG-CLASCOMP]

ABSTRACT: Classificatory reasoning involves the tasks of concept evaluation and classification, which may be performed with use of the strategies of concept matching and concept activation, respectively. Different implementations of the strategies of concept matching and concept activation are possible, where an implementation is characterized by the organization of knowledge and the control of information processing it uses. In this paper we define the tasks of concept evaluation and classification, and describe the strategies of concept matching and concept activation. We then derive the computational complexity of the tasks using different implementations of the task-specific strategies. We show that the complexity of performing a task is determined by the organization of knowledge used in performing it. Further, we suggest that the implementation that is computationally the most efficient for performing a task may be cognitively the most plausible as well.

27. A. Goel, J. Josephson and P. Sadayappan, Concurrency in Abductive Reasoning, appears in *Proceedings of the Knowledge-based Systems Workshop*, DARPA, April 21-23, St. Louis, Missouri, pages 86-92. [87-AG-CONCUR]

ABSTRACT: The information processing task in abductive reasoning is to infer a best explanation for a set of data. Some typical subtasks of this are generating hypotheses that can account for various portions of the data, and synthesizing a composite hypothesis that best explains the whole data set. In this paper we provide task-specific concurrent algorithms for some of the subtasks of abductive reasoning. In particular we present a blackboard architecture and a marker algorithm for the task of synthesizing

a composite hypothesis.

28. A. Goel, P. Sadayappan, J. Josephson and N. Soundararajan, Distributed Synthesis of Composite Explanatory Hypothesis, OSU CIS LAIR Technical Report [87-AG-DISSYNCOMP]. (Superseded by [88-AG-CONSYNCOMP].)
29. A. Goel, B. Chandrasekaran and D. Sylvan, JESSE: An Information Processing Model of Policy Decision Making, appears in *Proceedings of the Expert Systems in Government Conference*, 1987, October 19-23, Washington D.C., pages 178-187. [87-AG-JESSEESIG] (This is a revised and expanded version of an earlier paper entitled "JESSE: A Conceptual Model of Political Decision-Making", by A. Goel and B. Chandrasekaran, which appeared in MAICSS-87, Chicago, Illinois, April 24-25, 1987, pages 133-135.)

ABSTRACT: JESSE is an experimental system that models some aspects of Japanese energy policy decision making. It provides a functional architecture for decision making by the Japanese political and economic elite in the domain of her energy supply security. The system is initiated by supplying information about an energy-related event. It recognizes the threat posed by the event to Japanese energy supply security, and delivers a set of plans appropriate for the situation. In deciding on a set of plans, the system takes into account the state of Japanese foreign relations which impose constraints on the choice of policy options. JESSE is an integrated knowledge-based system. It contains multiple modules that perform the generic information processing task of Classification, and a module that performs the generic task of Plan Selection and Refinement. Each classification module is made up of a small number of problem solving agents that cooperatively accomplish the task of Classification. Similarly, the module for Plan Selection and Refinement contains a small number of cooperating planning agents. Thus, the complex information processing task of decision making is achieved collectively by an ensemble of problem-solving and planning agents acting in concert with one another.

30. J. Josephson, A Framework for Situation Assessment: Using Best-Explanation Reasoning to Infer Plans from Behavior, appears in *Proceedings of the Expert Systems Workshop*, DARPA, April, 1987, St. Louis, Missouri. [87-JJ-FRAME]

ABSTRACT: We propose a computational framework for battlefield situation assessment, describing how the relevant knowledge can be organized,

represented, and used in the service of problem solving. We describe how the reasoning processes can be controlled to avoid excessive and impractical amounts of search, and indicate how the computations can be distributed in a natural way to spread the burden over a community of separate processors and processing sites. This design extends previous work done at The Ohio State LAIR on diagnostic reasoning and representation of plan understanding, and applies it to a particular military information-processing problem, a species of the more general intellectual task of inferring plans and intentions from behavior.

31. D. Herman and K. Chevrier, Using DSPL in INTERLISP-D, OSU CIS LAIR Technical Report [87-DH-DSPLINTERLISP].

ABSTRACT: DSPL (Design Specialists and Plans Language) is a language developed for implementing expert systems which perform a kind of design problem solving. This document covers various details of loading and interacting with DSPL on a Xerox 1108 Lisp machine (a.k.a. Dandelion), running at least the Buttress release of LOOPS with at least the Koto release of INTERLISP-D. It is assumed that the reader is familiar with both LOOPS and INTERLISP-D on a Dandelion, as well as an exposure to the theoretical motivations underlying the DSPL Language.

32. D. Herman, The DSPL Manual, OSU CIS LAIR Technical Report [87-DH-DSPLMANUAL].

Three papers collected to assist the user in DSPL: [87-DH-DSPLINTERLISP], [87-DH-DSPLTUTORIAL] and [86-DB-KNOWCO].

33. D. Herman and D. Brown, DSPL: Language for Routine Design and Planning, appears over four issues of *Applied Artificial Intelligence Reporter*, April-July, 1987, Volume 4, Number 4-7, Pages 14-15, 14-15, 8-9, 15-19. [87-DH-DSPLTUTORIAL]

34. D. Myers, J. Davis and D. Herman, A Task Oriented Approach to Knowledge-Based System for Process Engineering Design, appears in *Computers in Chemical Engineering, special issue on AI in Chemical Engineering Research and Development*, January, 1988. [87-DM-TASKAPP] (Superseded by [88-DM-STILL].)

35. W. Punch and T. Bylander, CSRL: An Expert System Programming Language, a tutorial series appearing in *Applied Artificial Intelligence Reporter*, November and December of 1986 and January of 1987. [87-WP-AAIREP]

36. J. Josephson, B. Chandrasekaran, J. Smith and M. Tanner, A Mechanism for Forming Composite Explanatory Hypotheses. This paper is a revised and expanded version of an earlier paper by the same title which appeared in *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Casual and Strategic Aspects of Diagnostic Reasoning*, 1987, and which subsumed a paper entitled "Abduction by Classification and Assembly", presented at The Philosophy of Science Association Biennial Meeting for 1986 and appearing in *PSA* 1986, Vol. 1. This paper is OSU CIS Technical Report [88-JJ-MECHANISM]. It is included as Appendix F to this report.
37. B. Chandrasekaran, A. Goel and D. Allemang, Connectionism and Information Processing Abstractions, appears in *Proceedings of the AAAI spring symposium on Parallel Models of Intelligence: How Can Slow Components Think So Fast?*, Stanford University, March 1988, pages 66-85. [88-BC-CONPROAB] (A shorter version of this paper appeared in *Brain and Behavioral Science* as a commentary on Smolensky's paper "On the proper treatment of Connectionism".)

ABSTRACT: Since Connectionism challenges some of the basic assumptions on which much of Artificial Intelligence research has been based, it is important to examine the nature of representations and the differences between the Symbolic and Connectionist in this regard. Even though Symbolic and Connectionist systems may appear to yield the same functionality, we discuss how there is greater distinction between them than the Connectionist architectures being mere implementations of corresponding Symbolic algorithms. The two accounts differ fundamentally in terms of representational commitments, and thus in principle they offer alternative information processing theories. Nevertheless, we argue that the hard work of theory formation in Artificial Intelligence remains at the level of proposing the right information processing abstractions, since they provide the content of the representations. When, and if, we have Connectionist implementations solving a variety of higher level cognitive problems, the design of such systems will have these information processing abstractions in common with the corresponding Symbolic implementations. The information processing level specification of a theory of intelligence will then lead to decisions about which transformations on representation are best performed by means of Symbolic algorithms and which by Connectionists networks. In essence we claim that while Connectionism is a useful corrective to some of the basic assumptions of the Symbolic paradigm, for the most of the central issues of intelligence Connectionism is only marginally

relevant.

38. B. Chandrasekaran, Design: An Information Processing-Level Analysis, OSU CIS LAIR Technical Report [88-BC-DESIGN]. This is a draft version of Chapter 2 of the book *Design Problem Solving: Knowledge Structures and Control Strategies*, by D.C. Brown and B. Chandrasekaran. It is included as Appendix G to this report.
39. B. Chandrasekaran and A. Goel, From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task, appears in *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 18, No. 3, May/June 1988, pp. 415-424. [88-BC-FROMNO] Included as Appendix E to this report.
40. B. Chandrasekaran, "Deep" Models and Their Relation to Diagnosis, appears in *Artificial Intelligence in Medicine*, Volume 1, Number 1, 1989, pp. 29-40, Burgverlag, Tecklenburg, Germany. [88-BC-JAPAN] (It is a revised version of [87-BC-JAPAN].)

ABSTRACT: In this paper we distinguish between deep models in the sense of scientific first principles and deep cognitive models where the problem solver has a qualitative symbolic representation that accounts for how a system "works". We analyze diagnostic reasoning as an information processing task, identifying the generic types of knowledge (and reasoning) needed for the task to be performed adequately. If these are available an integrated collection of generic problem solvers can produce a diagnostic conclusion. The need for deep or causal models arises when some or all of these types of knowledge are missing in the problem solver. We provide a typology of different knowledge structures and reasoning processes that play a role in qualitative or functional reasoning and elaborate on functional representations as deep cognitive models for some aspects of causal reasoning in medicine.

41. B. Chandrasekaran, Generic Tasks As Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples, Appears in *Knowledge Engineering Review*, 1989, pp. 183-210. [88-BC-ROUTINE] (Supersedes [87-BC-HIERAR] and [87-BC-TOOLEV]). Included as Appendix D to this report.
42. K. Chevrier, M. Dejongh and O. Fischer, Using DSPL in KEE, OSU CIS LAIR Technical Report [88-KC-DSPLKEE].

ABSTRACT: DSPL (Design Specialist and Plan Language) is a language developed for implementing expert systems which perform routine design problem solving. This document covers various details of loading and interacting with DSPL in KEE. It assumes that the reader is familiar with KEE, and has an exposure to the theoretical motivations underlying the DSPL language.

43. A. Goel, J. Kolen and D. Allemang, Learning in Connectionist Networks: Has the Credit Assignment Problem been Solved?, OSU CIS LAIR Technical Report [88-AG-CONNETS].

ABSTRACT: A central problem of learning in an information processing system is that of credit assignment. This is usually constructed to be the problem of identifying which components of the system are responsible for incorrect system performance. In connectionist models this takes the form of adjusting the weights of connections between the processing units in the network in accordance with the generalized delta rule. The success of this scheme for learning in connectionist networks at modeling certain aspects of human behavior has led to the claim that the credit assignment problem has been solved. However, in this paper we suggest that the interpretation of the credit assignment problem on which this claim is based is inadequate both computationally as well as epistemically. From the computational perspective, a useful solution to the problem should be efficient so that learning can take place in reasonable time. From the epistemic viewpoint, the content of what is learned must be such that the system can perform a given task not only for the specific cases on which it was trained, but also for novel cases of the same task. We report on two experiments which suggest that the learning scheme based on the generalized delta rule ensures neither computational efficiency nor epistemic adequacy. We reluctantly conclude that an adequate, general solution to the credit assignment problem has not yet been found.

44. A. Goel, P. Sadayappan, and J. Josephson, Concurrent Synthesis of Composite Explanatory Hypotheses, appears in the proceedings of the Seventeenth International Conference on Parallel Processing, St. Charles, IL Aug 15-19 1988. [88-AG-CONSYNCOMP] (This paper supersedes [87-AG-DISSYNCOMP].)

ABSTRACT: The information processing task of abduction is to infer a hypothesis that best explains a set of data. A typical subtask of this is to synthesize a composite hypothesis that best explains the data set from

elementary hypotheses that can explain various portions of the data. In this paper, we present a computational model for concurrent synthesis of composite explanatory hypotheses that can be realized on a distributed memory, message passing, parallel machine. In this model, a process is associated with each datum to be explained as well as with each elementary hypothesis that can explain some portion of the data, and the control of processing alternates between the data and hypotheses' processes. In each cycle of processing, the data and the hypotheses' processes view the problem solving from their perspectives, and add to the growing composite explanatory hypothesis until a best explanation is synthesized. We analyze the time complexity of the concurrent algorithms, and discuss the architectural implications of the model.

45. A. Goel and B. Chandrasekaran, Integrating Model-based Reasoning and Case-based Reasoning for Design Problem Solving, appears in the *Proceedings of the AAAI Workshop on AI in Design*, St. Paul, Minnesota, August 1988. [88-AG-REASON]

ABSTRACT: Design is a complex information processing activity. What problem solving strategy is appropriate for the design of a specific artifact depends on what knowledge is available to the designer. One form in which design knowledge is often available is that of previously designed artifacts. For this reason, case-based reasoning is an attractive approach to design problem solving. A main issue in case-based design is how to adapt the structure of an existing design to achieve a new device functionality. This capability requires a causal understanding of how the structure of a device enables the accomplishment of its function. Such a causal understanding can often be expressed as a functional model of the device that specifies the designer's knowledge about the role of the various components and their relations in the functioning of the device. The Functional Representation scheme is a method for organizing and representing an agent's causal understanding of devices. In this scheme, the agent's understanding of the functioning of a device is expressed as behaviors that compose the functions of its structural components into device functions. In this paper, we illustrate how this organization of knowledge may enable a designer to identify the portions of the device structure that need to be modified to achieve a new functionality, and to reason about the effects of these structural changes. The integration of this capability with that of indexing, storing, and retrieving previous design cases can provide a powerful strategy for efficiently solving complex design problems.

46. J. R. Josephson, A Foundation for Task-Specific Explanation for Community-of-Minds Knowledge Systems, appears in the *Proceedings of AAAI-88 Workshop on Explanation*, August 22, 1988, St. Paul, Minnesota, pp. 87-89. [88-JJ-ENGRAMS]

47. John R. Josephson, Reducing Uncertainty by Using Explanatory Relationships, appeared in the *Proceedings of the Space Operations Automation & Robotics Workshop* sponsored by the USAF and NASA, Dayton, Ohio, July 1988. [88-JJ-REDUCING]

ABSTRACT: Explanatory relationships can be used effectively to reduce the uncertainty that remains after diagnostic hypotheses have been scored using local matching.

48. A. Keuneke and D. Allemang, Understanding Devices: Representing Dynamic States, OSU CIS LAIR Technical Report [88-AK-DYNSTATES].

ABSTRACT: An agent's understanding of a device is limited by the incompleteness of his representational model. Subsequent model-based reasoning can only be as good as the device model used. In order for the model of a device to be considered a cognitive model, the representation must provide for the organization and abstractions which cognitive agents bring to bear during problem solving. In this paper we suppose an agent understanding a device. We identify conceptual transitions of device behaviors that such an agent makes easily, and we argue for the necessity of a new type of abstraction for model representations in order to provide such capabilities. Since the abstraction is particularly useful for cyclic processes, we illustrate its usefulness in devices which possess cyclic behaviors.

49. A. Keuneke and D. Allemang, Exploring the No-Function-In-Structure Principle, appears in *Journal of Experimental and Theoretical Artificial Intelligence*, Volume 1, pages 79-89, 1989. [88-AK-EXPLOR]

50. D. Myers, J. Davis and D. Herman, A Task Oriented Approach to Knowledge-Based Systems for Process Engineering Design, OSU CIS LAIR Technical Report [88-DM-STILL]. An earlier version of this paper appears in *Computers in Chemical Engineering, special issue on AI in Chemical Engineering Research and Development*, January, 1988. (This paper supersedes [87-DM-TASKAPP].)

ABSTRACT: Expert systems for process engineering design applications provide a means of capturing not only calculations but also the decision-making knowledge and efficient problem-solving methods of the design

expert. Many important design applications in this domain involve design strategies and knowledge which are well structured. Our task-oriented approach recognizes this structure in the design task and exploits it by describing the design in terms of identifiable types of knowledge and a specific problem-solving strategy. DSPL is an expert system programming shell which allows knowledge characteristics of process engineering design problems to be explicitly represented according to the design task structure and offers an enhanced programming framework over first generation techniques emphasizing rule, frame, and logic levels. STILL, an expert system for the design of sieve tray distillation columns, provides an application of the DSPL language and a demonstration of the methodology.

51. Donald A. Sylvan, Ashok Goel, and B. Chandrasekaran, Analyzing Political Decision Making from an Information Processing Perspective: JESSE, to appear in the *American Journal of Political Science*. [88-DS-JESSE] (This paper supersedes [88-DS-JESSEPOLIT].)

ABSTRACT: Political behavior is argued to result from interacting political and information processing mechanisms. Political mechanisms deal with the values, interests, and influence of the political actors, while information processing mechanisms concern themselves with actors' use knowledge and experience in exploring the space of choices and actions.

Information processing approaches are argued to provide a language for expressing theories of political decision making with greater precision than before. This language also enables computational experimentation with the political theories. The general information processing approach is illustrated in detail for the specific domain of Japanese energy and foreign policy decision making. A political theory of decision making by the Japanese political and economic elite in the domain of her energy supply security is developed from an information processing perspective. This theory is embodied in an experimental system called JESSE. JESSE contains multiple modules that perform the generic task of Classification, and a module that performs the generic task of Plan Selection. The system is initiated by supplying information about an energy-related situation. It classifies the situation into types of threats posed to Japanese energy supply security and retrieves stored plans from memory. Its output can range from decision not to take any action, to a large number of actions some of which may seem contradictory.

The theory embodied in JESSE is argued to apply to political decision-making situations in which there are limits on institutional rivalry, the members of the decision making group have been socialized similarly, the problem domain is seen as sacrosanct, and there is substantial prior analysis and planning. The issue of validating the theory is addressed, and JESSE is found to be a plausible model of Japanese decision making in the domain of her energy supply security.

52. D. Sylvan, A. Goel and B. Chandrasekaran, An Information Processing Model of Japanese Foreign and Energy Policy Decision Making: JESSE. This paper was presented at the Annual Meetings of the International Studies Association, St. Louis MO, March 30- April 2, 1988 and was presented at the Seminar on Foreign and Defense Policy Decision Making, Mershon Center, The Ohio State University, January 27-28, 1988. This paper emphasizes the political aspects of the JESSE System. Another report in this collection, [87-AG-JESSEESIG], emphasizes the Artificial Intelligence design of the model. (Superseded by [88-DS-JESSE].)

53. S. Shum, J. Davis, W. Punch and B. Chandrasekaran, An Expert System Approach to Malfunction Diagnosis in Chemical Plants, appears in *Computers in Chemical Engineering*, Vol. 12 No. 1, 1988, pp.27-36. [88-SS-MALCHEM]

ABSTRACT: An efficient knowledge-based system approach to malfunction diagnosis in chemical processing plants is discussed. The approach involves a hierarchical diagnostic structure in which the nodes represent specific malfunction hypotheses. Instead of being a static collection of knowledge, the hierarchy is a collection of small individual specialists coordinated to arrive at an overall diagnosis. Each specialist contains compiled and qualitative domain knowledge for evaluating the malfunction hypothesis. This malfunction hierarchy is particularly effective in handling multiple symptom and multiple malfunction situations. Conclusions are based on a working prototype system which has built and tested to demonstrate the computational methodology.

54. M. C. Tanner and J. R. Josephson, Explanation and Abductive Justification, appears in *Proceedings of The Spring Symposium Series on Artificial Intelligence in Medicine*, Stanford University, March 22-24, 1988, pp. 95-96. [88-MT-EXPLANATION] (This paper supersedes [87-MT-ABJUSTIF].)

55. T. Bylander, D. Allemang, M. Tanner and J. Josephson, Some Results Concerning the Complexity of Abduction, appears in *Proceedings of The Spring Symposium Series on Artificial Intelligence in Medicine*, Stanford University, March 22,23,24, 1988, pp. 13-14. [88-TB-COMPLEXITY] (Superseded by [89-TB-COMPLEXITY]).

56. Tom Bylander, Dean Allemang, Michael C. Tanner, and John R. Josephson, Some Results Concerning the Computational Complexity of Abduction, appears in the *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, 1989, pages 44-54. [89-TB-COMPLEXITY] (This paper supersedes [88-TB-COMPLEXITY].)

ABSTRACT: The problem of abduction is to find the best explanation of a set of data or observations. In this paper we focus on one type of abduction in which the best explanation is the most plausible conjunction of hypotheses that explains all the data. We then present several computational complexity results demonstrating that very restrictive conditions must be satisfied for this type of abduction to be tractable (solving in polynomial time). Determining the plausibility and explanatory coverage of hypotheses must be tractable, there cannot be substantial incompatibility relationships or cancellation effects between individual hypotheses, and plausibility comparison between composite hypotheses must be logically weak.

57. David C. Brown and B. Chandrasekaran, *Design Problem Solving: Knowledge Structures and Control Strategies*, Pitman, London, Morgan Kaufmann, San Mateo, California, 1989.

ABSTRACT: In this book the 'classical' view of expert systems is contrasted with an alternative view in which a number of knowledge sources, each specializing in a *generic type* of knowledge-based reasoning, cooperate in a well-structured fashion to produce a solution. This view of expert problem solving encourages *knowledge-level* analysis of problems. A framework for design problem solving is presented that organizes the different types of knowledge and control strategies that are part of design activity. A detailed analysis of routine design is presented, using the design of a small air cylinder as an example. The knowledge used in the AIR-CYL expert system is captured in the DSPL language, which is specially designed for the expression of design knowledge. The book concludes with a research agenda for design problem solving.

58. Ashok Goel and B. Chandrasekaran, Use of Device Models in Adaption of Design Cases, to appear in the *Proceedings of the Second DARPA Workshop on Case-Based Reasoning*, Pensacola, Fla, May 31-June , 1989. [89-AG-NOCODE1]

ABSTRACT: A major issue in case-based design problem solving is how to adapt the structure of an existing design to achieve a similar but novel device function. The capability to suitably adapt retrieved design cases requires a causal understanding of how the structure of the device enables the accomplishment of its function. Such a causal understanding can often be represented as a functional model of the device that specifies the designer's knowledge about the role of the structural components and their relations in the functioning of the device. The Functional Representation scheme is a language for organizing and representing a problem solving agent's understanding of devices. In this scheme, the agent's understanding of a device is expressed as causal behaviors that compose the functions of its structural components into the device functions. We illustrate how this organization of knowledge enables a designer to efficiently identify the portions of the device structure that need to be modified to achieve similar but novel functions, and to effectively reason about the effects of these structural changes on the device functionality. The integration of this capability with that of retrieving and storing of design cases provides a computationally powerful strategy for solving non-trivial design problems.

59. Tom Bylander and Michael A. Weintraub, Integrating Qualitative and Associational Models in Diagnosis in Complex Domains, OSU CIS LAIR Technical Report [89-TB-INTMODELS].

ABSTRACT: Diagnosis is the task of explaining observations of a faulty system in terms of malfunctions and their causes. This task is difficult in complex domains, i.e., when the knowledge about the domain is incomplete, multiple faults exist, faults interact, and the system being diagnosed attempts to compensate for a malfunction. In complex domains, both associational and qualitative models have advantages and disadvantages for doing diagnosis. Associational models are efficient, but require a combinatorial number of associations to cover all possible situations. Qualitative models are effective, but require a combinatorial search of multiple fault hypotheses. This paper describes an architecture for diagnosis that attempts to extract the best of both models by using an abductive assembler to integrate an associational model with a qualitative model. Each subtask

of the abductive assembler is assigned to the model that can perform the subtask most efficiently and effectively.

60. B. Chandrasekaran, Michael C. Tanner, and John R. Josephson, Explaining Control Strategies in Problem Solving, appears in *IEEE Expert*, Spring 1989, pp. 9-24. [89-BC-EXPLAN] (Supersedes 87-BC-EXPCON.) Included as Appendix C to this report.
61. B. Chandrasekaran, What Kind of Computation is Intelligence? A Framework for Integrating Different Kinds of Expertise, to appear in the proceedings of the NASA Conference on Space Telerobotics, Pasadena, California, January 31-February 2, 1989. [89-BC-INTEGEXP]

ABSTRACT: The view that the deliberative aspect of intelligent behavior is a distinct type of algorithm, in particular a goal-seeking exploratory process using qualitative representations of knowledge and inference, is elaborated. There are other kinds of algorithms that also embody expertise in domains. We discuss the different types of expertise and how they can and should be integrated to give full account of expert behavior.

62. John R. Josephson, Diana Smetters, Richard Fox, Dan Oblinger, Arun Welch, and Gayle Northrup, Integrated Generic Task Toolset—Fafner Release 1.0—Introduction and User's Guide. OSU CIS LAIR Technical Report [89-JJ-FAFNER].

ABSTRACT: The Integrated Generic Task Toolset provides computational mechanisms especially suited for certain reasoning tasks which are typically encountered during the course of diagnosis, design, planning, and similar knowledge-based activities. By directly supporting a set of basic reasoning tasks, the toolset provides building blocks out of which more complex reasoning systems can be built. To support a particular reasoning task, the toolset provides a set of appropriate mechanisms, each with its own control strategies and characteristic ways of organizing and representing knowledge. Thus, the toolset provides to the knowledge system builder an advantage over rules, logic, semantic nets, frames, and neural nets, similar to the advantage that higher-level languages provide over assembly language for the computer programmer.

A release of the integrated toolset, called Fafner, is available for research use. Fafner provides the "toolbed" a framework for integrating the individual tools, the RA tool, and the CSRL tool. The RA tool provides the facilities for constructing recognition agents, agents designed to determine

the degree of match between a hypothesis and a situation. RA is the first of six tools to be included in this and later releases, and is an important foundation for the others. The CSRL tool builds on the foundation provided by RA, and allows the construction of hierarchies of classification specialists. Such hierarchies are used to perform the hierarchical classification task, an important part of both diagnosis and reasoning in general. Fafner is written completely in Common Lisp and CLOS (Common Lisp Object System) and is fully portable at that level.

63. John R. Josephson, A Layered Abduction Model of Perception: Integrating Bottom-up and Top-down Processing in a Multi-Sense Agent, to appear in the Proceedings of the NASA Conference on Space Telerobotics, Pasadena, California, January 31-February 2, 1989. [89-JJ-TOPBOTTOM] Included with this report as Appendix K.

64. Todd Johnson, Jack W. Smith Jr., and B. Chandrasekaran, Generic Tasks and Soar, OSU CIS LAIR Technical Report [89-TJ-GTSOAR].

ABSTRACT: This paper describes our research which is an attempt to retain as many of the advantages as possible of both task-specific architectures and the flexibility and generality of more general problem-solving architectures like Soar. It investigates how task-specific architectures can be constructed in the Soar framework and integrated and used in a flexible manner. The results of our investigation are a preliminary step towards unification of general and task-specific problem solving theories and architectures.

65. Mike Weintraub and Tom Bylander, QUAWDS: A Composite Diagnostic System for Gait Analysis, OSU CIS LAIR Technical Report [89-MW-QUAWDS].

ABSTRACT: QUAWDS is a system for analyzing human gait. QUAWDS integrates associational and qualitative models of knowledge into a diagnostic system, taking advantage of the tasks each kind of model can determine efficiently and effectively. An abductive assembler is used to coordinate the different models. The result is a diagnostic solution that is "locally best," i.e., no single change to the answer will produce a better solution. We believe QUAWDS' architecture is suitable for many complex domains.

66. Ashok Goel and B. Chandrasekaran, Functional Representation of Designs and Redesign Problem Solving, to appear in *Proceedings of the Eleventh*

International Joint Conference on Artificial Intelligence, Detroit, Michigan, Aug., 1989. [89-AG-NOCODE2]

ABSTRACT: The information processing task of redesign and its subtasks of diagnosis and repair are analyzed. Various kinds of knowledge required for redesign problem solving are identified, and a scheme for representing them is described. In this scheme, the functions of the device and its structural components are represented explicitly, and causal and anticipatory knowledge about its design is organized around these functions. This functional representation language also provides primitives for representing and accessing knowledge of domain principles such as Physics laws. The use of functional representation of designs in redesign problem solving is illustrated for the redesign of the reaction wheel assembly aboard the Hubble space telescope.

DISTRIBUTION LIST

addresses	number of copies
RADC/COES ATTN: Robert N. Ruberti Griffiss AFB NY 13441-5700	5
Ohio State University Lab for AI Research Dept of Computer & Info Science 1314 Kinnear Road Columbus OH 43212	5
RADC/DOVL Technical Library Griffiss AFB NY 13441-5700	1
Administrator Defense Technical Info Center DTIC-FDA Cameron Station Building 5 Alexandria VA 22304-6145	5
Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington VA 22209-2308	2
AFCSA/SAMI ATTN: Miss Griffin 10363 Pentagon Washington DC 20330-5425	1
HQ USAF/SCTT Pentagon Washington DC 20330-5190	1
SAF/AOSC Pentagon 4D-267 Washington DC 20330-1000	1

Director, Information Systems
OASD (C3I)
Room 3E187
Pentagon
Washington DC 20301-3040

1

HQ AFSC/XTKT
Andrews AFB DC 20334-5000

1

HQ AFSC/XTS
Andrews AFB MD 20334-5000

1

HQ AFSC/XRK
Andrews AFB MD 20334-5000

1

HQ SAC/SCPT
OFFUTT AFB NE 68113-5001

1

DTESA/RQE
ATTN: Mr. Larry G. McManus
Kirtland AFB NM 87117-5000

1

HQ TAC/DRIY
ATTN: Mr. Westerman
Langley AFB VA 23665-5001

1

HQ TAC/DOA
Langley AFB VA 23665-5001

1

HQ TAC/DRCA
Langley AFB VA 23665-5001

1

ASD/AFALC/AXAE
ATTN: W. H. Dungey
Wright-Patterson AFB OH 45433-6533

1

WRDC/AAAI
Wright-Patterson AFB OH 45433-6533

1

AFIT/LDEE
Building 640, Area B
Wright-Patterson AFB OH 45433-6583

1

WRDC/MLTE
Wright-Patterson AFB OH 45433

1

WRDC/FIES/SURVIAC
Wright-Patterson AFB OH 45433

1

AAMRL/HE
Wright-Patterson AFB OH 45433-6573

1

2750 ABW/SSLT
Building 262
Post 11S
Wright-Patterson AFB OH 45433

1

AFHRL/OTS
Williams AFB AZ 85240-6457

1

AUL/LSE
Maxwell AFB AL 36112-5564

1

HQ Air Force SPACECOM/XPYS 1
ATTN: Dr. William R. Matoush
Peterson AFB CO 80914-5001

Defense Communications Engr Center 1
Technical Library
1360 Wiehle Avenue
Reston VA 22090-5500

C3 Division Development Center 2
Marine Corps
Development & Education Command
Code DIOA
Quantico VA 22134-5080

US Army Strategic Defense Command 1
DASD-H-MPL
PO Box 1500
Huntsville AL 35807-3801

Commanding Officer 1
Naval Avionics Center
Library
D/765
Indianapolis IN 46219-2189

Commanding Officer 1
Naval Ocean Systems Center
Technical Library
Code 96428
San Diego CA 92152-5000

Commanding Officer 1
Naval Weapons Center
Technical Library
Code 3433
China Lake CA 93555-6001

Superintendent 1
Naval Post Graduate School
Code 1424
Monterey CA 93943-5000

Commanding Officer 2
Naval Research Laboratory
Code 2627
Washington DC 20375-5000

Space & Naval Warfare Systems COMM 1
PMW 153-3DP
ATTN: R. Savarese
Washington DC 20363-5100

Commanding Officer 2
US Army Missile Command
Redstone Scientific Info Center
AMSMI-RD-CS-R (Documents)
Redstone Arsenal AL 35898-5241

Advisory Group on Electron Devices 2
Technical Info Coordinator
ATTN: Mr. John Hammond
201 Varick Street - Suite 1140
New York NY 10014

Los Alamos Scientific Laboratory 1
Report Librarian
ATTN: Mr. Dan Baca
PO Box 1663, MS-P364
Los Alamos NM 87545

Rand Corporation 1
Technical Library
ATTN: Ms. Doris Helfer
PO Box 2138
Santa Monica CA 90406-2138

USAG 1
ASH-PCA-CRT
Ft. Huachuca AZ 85613-6000

1839 EIG/EIET 1
ATTN: Mr. Kenneth W. Irby
Keesler AFB MS 39534-6348

JTFPO-TD 1
Director of Advanced Technology
ATTN: Dr. Raymond F. Freeman
1500 Planning Research Drive
McLean VA 22102

HQ ESC/CWPP 1
San Antonio TX 78243-5000

AFEWC/ESRI
San Antonio TX 78243-5000

3

485 EIG/EIR
ATTN: M Craft
Griffiss AFB NY 13441-6348

1

ESD/XTP
Hanscom AFB MA 01731-5000

1

ESD/ICP
Hanscom AFB MA 01731-5000

1

ESD/AVSE
Building 1704
Hanscom AFB MA 01731-5000

2

HQ ESD SYS-2
Hanscom AFB MA 01731-5000

1

Director
NSA/CSS
T513/TDL
ATTN: Mr. David Marjarum
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
W166
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R24
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R21
9800 Savage Road
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
DEFSMAC
ATTN: Mr. James E. Hillman
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R5
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R8
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
S21
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
W07
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
W3
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R523
Fort George G. Meade MD 20755-6000

2

AFHRL/LRG
ATTN: Mr. M. Young
WPAFB OH 45433-6503

1

AAMRL/HEA
ATTN: Dr. B. Tsou
WPAFB OH 45433-6503

1

AAMRL/HED
ATTN: Maj M.R. McFarren
WPAFB OH 45433-6503

1

WRDC/KTD
ATTN: Dr. D. Hopper
WPAFB OH 45433-6503

1

AFIT/ENG
ATTN: Maj P. Amburn
WPAFB OH 45433-6053

1

CEETL-GL-VT
ATTN: Mr. T. Jorgensen
Ft Belvoir VA 22060-5546

1



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.